MASTER IN COMPUTER SCIENCE

Enforcing foreign key constraints in legacy systems

Carl, Henry; Staelens, Thibaud

Award date:
2018

Awarding institution:
University of Namur

Université de Namur
Faculty of Computer Science
Academic Year 2017–2018

Enforcing foreign key constraints in legacy systems

Carl HENRY Thibaud STAELENS

Internship mentor: Jens WEBER

Supervisor: _____ (Signed for Release Approval - Study Rules art. 40)
Anthony CLEVE

A thesis submitted in the partial fulfillment of the requirements
for the degree of Master of Computer Science at the Université of Namur

Acknowledgement

This thesis is based on the work produced during a three-and-a-half month internship in Victoria, Canada. There we integrated a laboratory that works on several subjects for improving health-care information systems all over Canada. A part of this laboratory uses real large electronic medical record (EMR) software systems for effectuating research on the management of technical debt in order to facilitate database schema evolution. During our internship we used the OSCAR EMR as a case study to progress in that area.

We would like to thank our supervisor, Anthony CLEVE, for offering us the possibility to accomplish an internship abroad full of happy experiences and discoveries in Canada, for his support, advices, discussions and enlightenment on the subject of database.

We would like to thank our internship mentor, Jens WEBER, for his support, advices and help during our internship and we also would like to thank his sympathetic research team members for welcoming us among them.

Finally, we would like to thank our relatives - parents, friends, colleagues - for their support and their help during the writing of this thesis.

Abstract

The world in which we live is constantly changing and information systems must also evolve to cope with these changes. However, legacy systems, which are very large and complex, cannot easily evolve. In particular, the semantics of the logical database schema of these systems is often not documented. Therefore, the conceptual schema will have been lost over time. Unimplemented foreign keys are then lost, making the task of upgrading this database perilous. To compensate for this, this conceptual schema must be restored. Database reverse engineering provides tools for retrieving this lost information. Nowadays, locating implicit foreign keys in order to implement them is no longer a problem. However, the restoration of these relational constraints is not yet a subject of much study.

This thesis aims at proposing solutions to facilitate the implementation of these lost implicit foreign keys thanks to a method of evaluation of the risk of implementing a constraint through a calculation of the schema transformations necessary to its addition as well as the probability and the importance of the errors that it could generate at the application level.

Résumé

Le monde dans lequel nous vivons est en constante évolution et les systèmes d'information se doivent d'évoluer eux-aussi pour faire face à ces changements. Cependant, les « legacy systems », très grands et très complexes, ne peuvent pas facilement évoluer. Particulièrement, la sémantique du schéma logique de la base de données de ces systèmes n'est bien souvent pas documenté ce qui entrainera une perte du schéma conceptuel au fil du temps. Les clés étrangères non implémentées sont alors perdues, ceci rendant périlleuse la tâche de faire évoluer cette base de données. Pour pallier cela, ce schéma logique doit être restauré. La rétro-ingénierie de base de données fournit des outils pour retrouver cette information perdue. A l'heure actuelle, localiser des foreign keys implicite en vue de les implémenter n'est plus un problème. Cependant, la restauration de ces contraintes relationnelles n'est pas un sujet d'étude encore beaucoup entrepris.

Cette thèse vise à proposer des solutions pour faciliter l'implémentation de ces foreign keys implicites perdues grâce à une méthode d'évaluation du risque d'implémentation d'une contrainte à travers un calcul des transformations de schéma nécessaire à son ajout ainsi que de la probabilité et l'importance des erreurs que cela pourrait engendrer au niveau applicatif.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, new developed information systems are becoming increasingly rare. Indeed, developers will tend to adapt already developed systems to meet their requirements instead of developing new systems from scratch.

Year	New projects	Enhancements	Repairs	Total
1950	90	3	7	100
1960	8,500	500	1,000	10,000
1970	65,000	15,000	20,000	100,000
1980	1,200,000	600,000	200,000	2,000,000
1990	3,000,000	3,000,000	1,000,000	7,000,000
2000	4,000,000	4,500,000	1,500,000	10,000,000
2010	5,000,000	7,000,000	2,000,000	14,000,000
2020	7,000,000	11,000,000	3,000,000	21,000,000

Figure 1.1: Work distribution of system developers (taken from [21])

Figure 1.1 shows that estimations for 2020 anticipate that only 30% of system developers will work on new projects while 70% will work on enhancements and repairs.

As the world is in constant evolution, these systems have to adapt to it so that they are also in continuous evolution. This leads to huge and very complex systems that are called legacy systems.

1.1 Legacy System

A legacy system is a large information system in which the source code and the architecture is the product of a long maintaining that can cause some obsolescence, incompleteness and inconsistencies but whose replacement cost would be too high and risky. For that type of systems, the most reasonable solution is to incrementally update them, fixing specific parts each time. But doing that

is not an easy task considering that generally these systems lack in documentation, reference persons that know them well and who can be reached. Moreover, ordinarily, the technologies involved are old and so can slow down the whole system.

But, if someone is researching the subject of legacy system, it will rapidly appear to him that there are a lot of other denominations that, even if they are quite straightforward, need to be mentioned: legacy system, legacy software, legacy hardware or legacy database.

1. Legacy Software : Has the same characteristics as a legacy system but only at the software level.
2. Legacy Hardware : Has the same characteristics as a legacy system but only at the hardware level. In this thesis this term will never be used because there will never be any hardware level discussion.
3. Legacy Database : Has the same characteristics as a legacy system but only at the database level.

Even if this thesis is essentially focusing on databases, it will generally be question of **legacy systems** instead of legacy databases because this first term is more general and includes every part of an information system.

1.2 Legacy system evolution

As said before, information systems are in constant evolution and obviously so are legacy systems. Generally, the software developers tend to evolve only the software part of the system while the database has to evolve as well. To do that, a good understanding of the database is required and in that perspective documenting a database and specifically specifying the foreign keys is very important. If it is not the case, with time, the initial conceptual schema can be lost, mistakes can be made on the data representation, non-specified relational constraints can be violated and the data integrity can be lost. It is very difficult and dangerous to perform a database evolution on an erroneous schema or with data integrity violations.

But trying to correct the database schema and to restore this data integrity is not a trivial task. This thesis is focusing on the restoration of relational constraints that were lost.

1.3 Implicit Foreign Keys

In a relational database, the greatest way to link tables together and so bringing additional information to the logical schema is the mechanism of the foreign key.

A foreign key is *a column or group of columns in a relational database table that provides a link between data in two tables. It acts as a cross-reference between tables because it references the primary key of another table, thereby establishing a link between them*[6].

In practice, it can express a lot of different semantic patterns and it is not always easy to correctly interpret and understand the conceptual construct it represents.

Moreover, most of the time in legacy systems a lot of foreign keys are not documented or implemented. The reasons for that are multiple : laziness from the database developer, lack of time, better performance, ... But however, in most cases the software has been built with the idea that there should be relational constraints in the database. So the case of these missing foreign keys that are however present in the logical schema of the database are called "implicit foreign keys".

1.4 Reverse engineering and Implicit foreign key detection

One of the most important issues we are facing with legacy systems is that the lack of database documentation induces losses of relational constraints. How can we implement these implicit foreign keys if we are not aware of their existence because of the missing documentation?

Advanced methods of database reverse engineering can be used to retrieve the needed information about these foreign keys. The technology of database reverse engineering is for long time used for retrieving database documentation. However, the techniques for retrieving missing foreign keys is relatively new.

Nevertheless, the detection of these implicit foreign keys is not the concern of this paper (it will be briefly presented in chapter 2) but it is obvious that this step is crucial on the objective of implementing them. So this thesis has been made with the assumption that retrieving the list of the implicit foreign keys in a given database is not an issue.

1.5 Research question

1.5.1 Research subject

The main goal of this master thesis is to propose solutions to ease the implementation of missing implicit foreign keys of a database in a legacy system context. This process is called **enforcing foreign keys**[1]

1.5.2 Why is it important and difficult?

As said before, it is estimated that in a few years, the main jobs of computer scientists will be to maintain existing systems - and so large legacy systems.

Therefore, research in facilitating and improving the evolution of that kinds of systems is becoming more and more crucial. Moreover, this implicit foreign keys concern had not been the center of much research yet while any contribution in this area is needed.

Indeed, legacy systems evolution often requires database migration which cannot be safely done with unknown implicit foreign keys in this database.

But implementing those implicit foreign keys is a difficult problem.

[1]The detailed meaning of this term will be explained in section 3.4

First, there can be a lot of issues at the database level. That is to say : mismatching types between the foreign key column and the destination column, mismatching values, cascade foreign keys, ... All these problems have to be detected and solved before the addition of the foreign key.

Unfortunately, issues will not only rise at the database level but at the application level as well. Indeed, nothing can ensure that an application using the database will still be functional after this addition. A short example would be an application that never violates the logic of an implicit foreign key but that wrongly enters the data - enters a value in the foreign key column before entering the same value in the destination column for instance. Again, solutions to these application issues will have to be provided.

1.5.3 How can we enforce these foreign keys?

As we are facing two different but linked problems (database transformation and its propagation in the program), we will have to treat them separately for each step of the design and implementation of the solutions.

For the database transformation part, the idea is to develop a tool that will automatically take a list of missing foreign keys, treat them one by one and execute all the required transformations.

For the application propagation part, two directions can be explored :

1. Static Analysis
2. Dynamic Analysis

The static analysis would extract queries from the code. This extraction should spot us the areas of the code that can be problematic for a certain foreign key. In summary it will help us to find foreign keys *critical sections*.

The dynamic analysis is constituted of a mechanism called *trigger logger* that will help us detecting critical foreign keys.

The global goal of this thesis is to provide a tool that encompasses the database transformation tool and both the static and dynamic analysis in order to provide the user with a good visualization of the difficulty of enforcing a given foreign key so that he can decide which set of foreign keys he wants to enforce.

1.6 Thesis Structure

Chapter 2 is the state of the art related to the context in which this thesis was conducted, the database reverse engineering process, several tools that are used for our solutions and inspiring papers.

Chapter 3 states the research problem in more details than in this short introduction.

Chapter 4 describes the methodology we used to face this problem and explains decisions we had to make.

Chapter 5 details the algorithms or other design solutions we built according to this methodology.

Chapter 6 gives explanations about the implementation of the tools we developed based on these design solutions.

Chapter 7 presents the OSCAR system and discusses the results of our most advanced tool in this case study.

And finally, chapter 8 concludes this thesis by giving a summary of it, discussing the potential issues of our solutions and presenting the future works that we think are needed to improve these solutions.

Chapter 2

State of the art

First of all, it is important to note that there are not a lot of previous works on the precise subject of the enforcement of missing implicit foreign keys and the propagation impacts of these additions at the application level. Nevertheless it seemed primordial to present the context in which this thesis was conducted and to describe several works that had been undertaken on the border of this subject but whose results will be used to develop our specific solution.

This thesis is part of a collaborative project between the University of Namur and the University of Victoria whose objective is the advancement in the field of reducing technical debt in critical software. To that end, managing technical debt in database schemas of critical software is key.

In this state of the art, a paper [27] concerning the management of the technical debt will be first introduced.

Then, an interesting tool for generating up-to-date database documentation will be shortly presented. After that, an introduction to the field of database reverse engineering will be proposed in order to better understand the different developed tools used in this thesis that will be briefly presented and explained afterward - these tools being an implicit foreign key detection tool and a critical query extractor tool.

Finally, there will be a short discussion about a paper describing a solution that inspired the mechanism of "trigger logger" that will be presented in more details later.

2.1 Managing the technical debt

First of all, the term **Technical Debt** is a metaphor that is used to quantify the issues that can potentially arise from software evolution and maintenance actions that were undertaken in order to modify the functionality or behaviour of a system. In reality, the term technical debt can define a very large range of definition and even if we will stick here to the previously given definition, we encourage the reader to consult an interesting article [22] on this subject if s/he is interested.

Usually, this metaphor is used in a software program code or architecture context and not in the context of database applications (and even less regarding

database schemas). This distinction is important because these two technical debts are different mainly due to the fact that database schemas are linked to data instances which are not under control and often not accessible to software developers. *Weber, Cleve, Meurice and Ruiz* [27] are concentrating on a particular type of database schema related technical debt : missing referential integrity constraints or more commonly known as **foreign keys**. The authors argue that it should be considered as technical debt because of its importance for preserving data validity and quality in relational databases and because it allows complex transactions that can restore data integrity if violations had been made.

They describe two ways for this particular type of technical debt to arise [27] :

1. Because of incremental modifications of the database application's data model :

   This case can be the consequence of a difference in the view between application programmers who see the database only as a necessity that is generated and used by the application code which is the core of the system and database engineers who see the application code only as peripheral functions.

   This particular point of view from application programmers leads them to modify the application's data model without applying the according changes to the database layer.

2. Because of architectural changes (like a platform migration) :

   This case refers for example to a database migration from a noSQL database[1] to a relational database or upgrading an old version of DBMS to a new version that supports the foreign key monitoring. Both these situations result in increased technical debt because of the occurrence of implicit constructions[2].

The idea of reducing the technical debt was the genesis of our thesis and enforcing foreign keys enables this reduction, which is the focus of the thesis.

2.2 Automatically generating up-to-date database documentation

Reducing technical debt is very important in order to enable database schema evolution, but it would be better if this technical debt is as few as possible from the beginning.

[1] noSQL databases are not presented in this thesis, if the reader is interested to learn more about them we advise her/him to consult the publication *Exploring the merits of nosql* by *B. Jose and S. Abraham*[20].

[2] A debate can be raised on this point because it is not trivial to consider that some technical debt is **created** in these situations or if this technical debt was already there before. The reader can find the full thinking in *Managing Technical Debt in Database Schemas of Critical Software*[27].

A research conducted by *M. Linares-Vásquez, B. Li, C. Vendome and D. Poshyvanyk* focused on the idea of creating a process for *automatically generating always up-to-date natural language descriptions of database operations and schema constraints in source code methods*[23].

Their main motivation was that recent studies[3] demonstrated that handwritten comments in the database - that in theory could give to database developers almost sufficient information for having a satisfactory level of knowledge of the database - are generally not maintained or updated when the database source code changes[16][17].

They designed **DBScribe**, a tool that can help developers to understand how features are implemented in the database and - the most interesting part for this thesis - understand schema constraints that need to be satisfied.

DBscribe would be very useful for documenting relational constraints so that it would be easy to locate all foreign keys in a database, implicit or not.

This is a very interesting way of reducing technical debt by preventing generating it but as this thesis focuses on enforcing implicit foreign keys on legacy systems that already generated this technical debt, we turned ourselves towards another tool that uses database reverse engineering for locating missing implicit foreign keys. For this reason, the functioning of this tool is not detailed here. But if the reader is interested, we advice reading the related paper [23].

2.3 Database reverse engineering

According to *Chikofsky and Cross* [10], *Reverse engineering is the process of analyzing a subject system to:*

1. *Identify the system's components and their interrelationships.*
2. *Create representations of the system in another form or at a higher level of abstraction.*

But this definition is also valid for database reverse engineering.

And specifically, in the context of legacy systems where documentation is often lost, database reverse engineering is used to retrieve the logical schema of a database from its physical schema alone, in order to facilitate system maintenance, re-engineering, extension, migration or integration (this is a non-exhaustive list - the reader can find a more complete list in the book *Introduction to Database Reverse Engineering* by *J-L. Hainaut*[19]).

A valid view of database reverse engineering is to consider it simply as the reverse of the database forward engineering process [8][4] .

So let us define what database forward engineering is. It is the process that moves from the initial requirements of the business to the implementation code of the database through logical and physical design. Figure 2.1 summarizes the database forward engineering processes.

[3]studies from 2007 and 2009.

[4]According to *J-L. Hainaut*[19], database reverse engineering is more complex than that, but the author still qualifies this view as **a good starting point**.

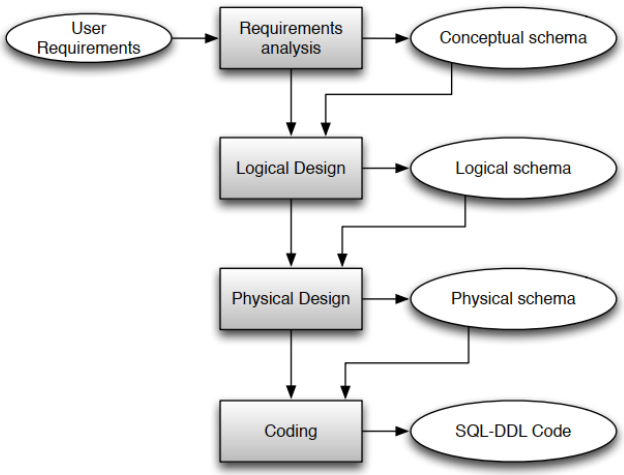


Figure 2.1: The database forward engineering processes (taken from [25])

These processes can be subdivided in subprocesses that will enhance the database quality. For instance, during the requirements analysis process, a normalization step that will improve readability, normality or minimality can be performed. Also, in the logical design phase, an optimization step can be achieved to improve performance. More subprocesses are described by *Hainaut* [19].

Now that we understand better which processes are required in database forward engineering, it is easier to describe database reverse engineering because its processes will be essentially the same as in database forward engineering but inversed.

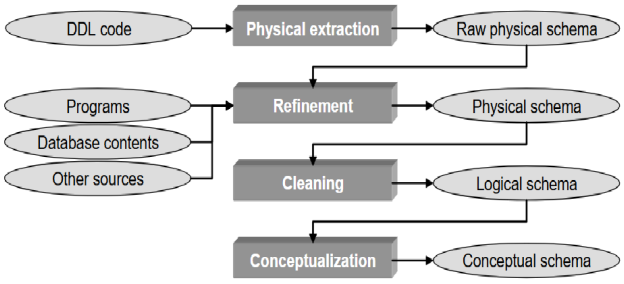


Figure 2.2: The database reverse engineering processes (taken from [13])

Figure 2.2 shows the different required processes in database reverse engineering.

Here are short descriptions of each of these processes (these descriptions are taken from [13])

1. Physical extraction :

   Parsing the DDL code for extracting the raw physical schema of the database.

2. Refinement :

   Refining the raw physical schema with additional structures and constraints found by analyzing other sources like Static program analysis[24], dynamic program analysis[18], evolution history analysis[14], ...

3. Cleaning :

   Removing the physical constructs in order to produce the logical schema.

4. Conceptualization :

   Deriving the conceptual schema implemented by the previously produced logical schema.

With Figure 2.3 taken from [19] we can appreciate the parallel between database forward and database reverse engineering. The symbol **inv** on the forward/reverse correspondences means that each process is the inverse of the other, while the symbol = indicates they are the same.

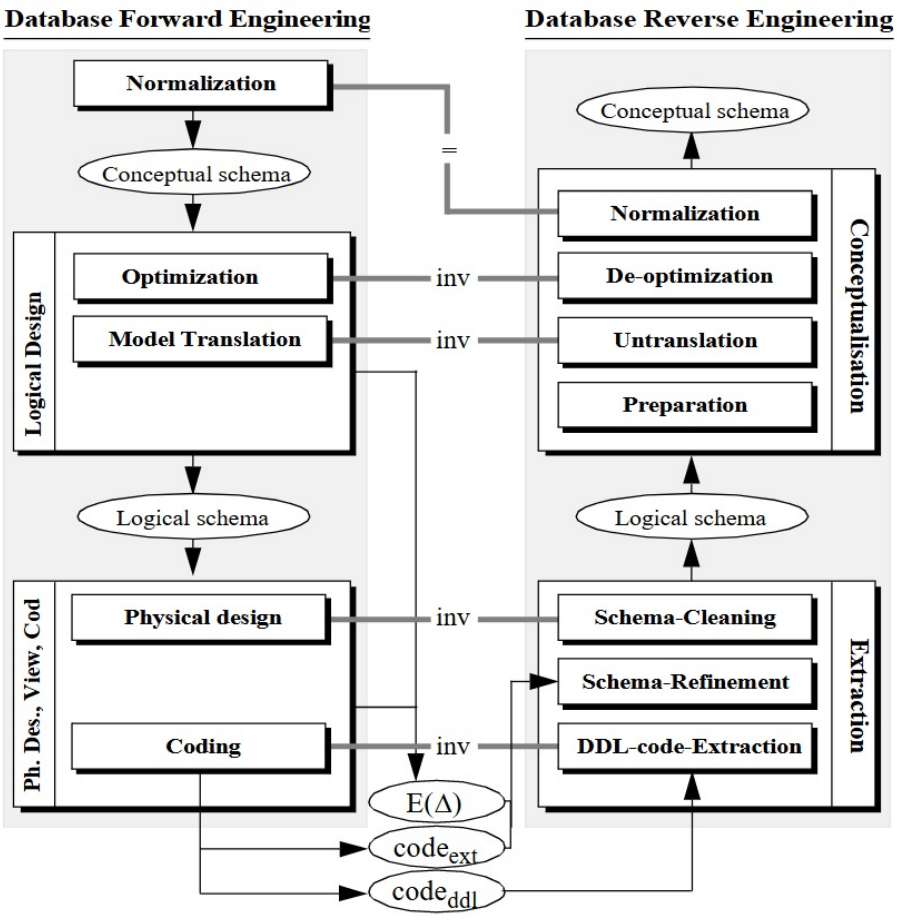


Figure 2.3: The database reverse engineering processes as the inverse of forward processes (taken from [19]).

As the purpose of this section is to give a quick introduction to what database reverse engineering is, we will not detail further the different processes.

2.4 Implicit foreign key detection tool

As explained in subsection 1.5.1, we are only focusing on the implementation of implicit foreign keys and its impacts. We considered that deciding where in the database a foreign key should be added is not part of this research. Luckily, this precise subject has already been researched by a team of two students of the University of Namur during an internship at the University of Victoria - see *Gobert and Maes* [25]. The two students have indeed developed a tool able to provide a list of all the foreign keys spotted as "potential implicit foreign keys" by its algorithm .

The purpose of the research of these two students wasn't only to provide the list we need. The main goal was to propose solutions to improve the database

comprehension of legacy systems through revisited "Database reverse engineering". A part of that process is indeed to retrieve the missing foreign keys. That particular process was divided in two steps.

1. Finding the missing foreign keys :

   This can be achieved through heuristics based on column names, SQL statement analysis, program slicing, DDL code analysis, etc and a new ORM analysis they developed. All these techniques can help to elaborate a list of candidate foreign keys.

2. Validating these candidate foreign keys :

   Gobert and Maes say it can be made by analyzing the data, the data usage, the technical constructs and especially by analyzing the historical schema of the database. They give the specific example of a candidate foreign key which its target column had been created after its source table. In this situation the candidate cannot be a foreign key and should then be eliminated from the list.

To help the reader understand their retrieving foreign key process, here is a simple schema they provided to synthesize it.

Figure 2.4: Fk detection method

They tested this tool on the case study OSCAR (which will be described in detail in chapter 7). With the first step, they found 440 candidate foreign keys that needed to be validated. Unfortunately, the second step wasn't fully achieved. They wanted to use a tool called "Key analysis" developed by Rever[4], but due to a lack of data in time, they managed to reject only a few foreign keys. On the other hand, their technique for analyzing the historical schema of the database permitted to reject 26 foreign keys.

The consequence of this is that they cannot provide 100% reliable candidate foreign keys. Nevertheless, this tool has been a great help to us for testing our proper tool.

2.5 DAHLIA

Another domain useful for us and already been researched is the area of the static analysis of the code of an application to seek where critical queries have been made. Obviously, it would be very interesting for our solution to be able to detect the parts of the code that cannot longer work after the modifications it made on the database.

A team of researchers from the University of Namur developed a tool called "DAHLIA" for Database ScHema EvoLutIon Analysis and it is described by *C. Nagy, L. Meurice, and A. Cleve* in the publication *Where was this SQL query executed? a static concept location approach* [26] as an "interactive and visual analyzer of database schema (and usage) evolution".

So its main purpose is to visualize particular schema versions of a database in 2D or 3D and it allows also to compare two schema versions.

But what interested us is the advanced version of DAHLIA (DAHLIA++) which its main concern is the "static extraction and analysis of the database accesses in the java source code" and so to be able to answer the question "where was a given erroneous SQL query executed in the source code?".

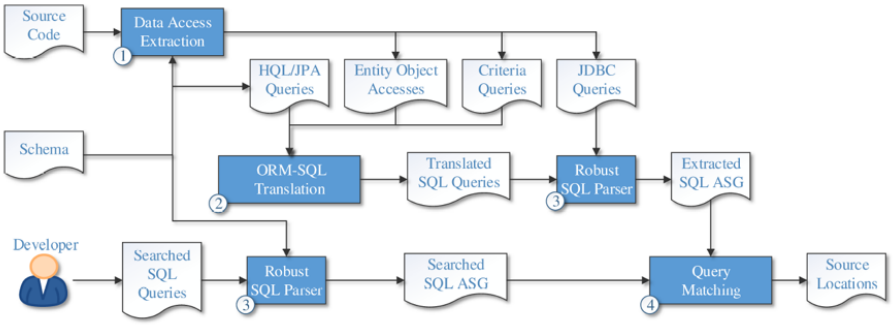


Figure 2.5: DAHLIA method for source location

Figure 2.5 shows all the steps the researcher developed through DAHLIA for answering that difficult question of the erroneous query localization in the source code.

For simplification purposes this process can be divided in four steps :

1. The extraction :

   Extracting the SQL queries, the HQL queries from hibernate and the JPQL queries from JPA.

2. The parsing :

   Parsing these extracted queries in order to construct the Abstract Semantic Graph.

3. The analysis :

   Analyzing some metrics, the columns and table access, the coding rules violation, ...

4. The matching :

   Trying to match the extracted query from the Java source code and the initial problematic query.

The profound complexity of this method will not be described here as it is not the subject of this thesis but here are some basic and simple explanations.

2.6 At the origin of trigger logger

Finally, let's describe a paper by *R. Balzer* [7] that greatly inspired the "trigger logger" technique that will be documented later.

First of all, nearly all constraints and hypotheses have exceptions. To deal with such exceptions there are three possibilities :

1. Simply not declare the constraint or hypothesis.
2. Not apply the constraint or hypothesis. That is to say treating it like a commentary so that the constraint is still documented.
3. Weaken the constraint or hypothesis to authorize the exceptions.

Most of the time option 1 or 2 is chosen and problems are found by steady tests algorithms. These problems are then fixed by hand. But in reality, only the third option is valid. The key is to formally declare the constraints but at the same time to authorize violations of these constraints.

They described two implementations to identify particular data that violate a constraint. The first one called "Paired Maintenance Constraint" interested us greatly.

The idea is to replace the original constraint by a couple of constraints called "guards" that assure that when there is a violation, a "Pollution Marker" is inserted to notify the violation and when the violation disappears the "Pollution Marker" is removed.

Thanks to the work of *R. Balzer* [7] the idea of tolerating inconsistencies emerged in our proper work.

Moreover, as said before, this work inspired us on the mechanism of "trigger logger" very similar to these "guards". Such mechanism is described further in this thesis.

Chapter 3

Problem statement

3.1 The Foreign key constraint

In order to understand what an implicit foreign key is, it is primordial to define a simple foreign key. A foreign key is a relation between two concepts. For example, a link between a car and its owner. In UML language, it can be represented as :

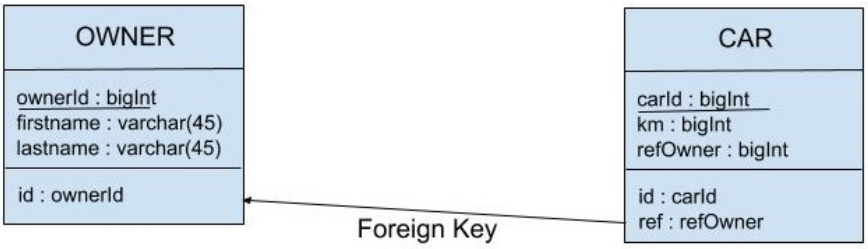


Figure 3.1: Classic Foreign Key Example

In Figure 3.1 the two concepts CAR and OWNER are represented as tables in a database and the foreign key is the relation that can associate a CAR with its OWNER.

In its essence, the foreign key mechanism is simply the representation of a particular type of logical constraint : a reference constraint. But in practice, this constraint can be used to express many different conceptual constructions on the logical schema. A non-exhaustive classification had been made by *A. Cleve* [12]. Nevertheless, as we are not interested here on the conceptual construction of a foreign key but mainly on its physical construction (is the foreign key implemented in the database or not?), we will not dwell on this part.

For simplification purpose, only the simplest physical construction of a foreign key like Figure 3.1 will be discussed in this thesis. In particular, the multi-column foreign keys will be avoided although it is not a rare construction to find

in databases. We argue that taking that case into consideration would greatly complicate our thinking on the problem whereas including it once the basic foreign key problem has been solved should not require too much time and many other resources.

Additionally, it is interesting to point out that considering only these basic foreign keys does not exclude the case of cascade foreign keys.

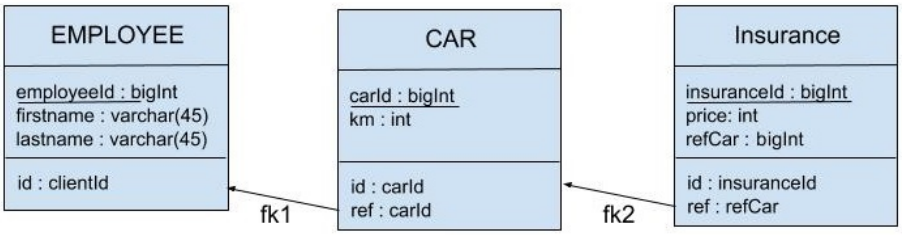


Figure 3.2: Cascade Foreign Key Example

In figure 3.2 if there are modifications on the first constraint, the second could be affected as well[1]. For instance, if the type of employeeId is changed to *varchar*, the type of carId must also be modified to *varchar* because of *fk1*. But because of *fk2* the type of refCar should also be changed. This is a complication that needs to be faced.

3.2 Explicit and implicit foreign keys

In the previous section, we defined the foreign key construction. Now, we will classify the different ways of expressing that construction. Two main ways of expressing a foreign key can be considered :

1. Explicit foreign key :

   Using DBMS integrated mechanisms by implementing the foreign key in the DDL code of the application. Using the MySQL example, there are two ways of implementing it :

   (a) declaring it when declaring the source table :

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    CONSTRAINT FK_PersonOrder FOREIGN KEY (PersonID)
    REFERENCES Persons(PersonID)
);
```

[1]Here, carID has the same value that the employeeID of the employee with whom the car is associated.

(b) adding it to a table after that table has already been declared :

```
ALTER TABLE Orders
ADD CONSTRAINT FK_PersonOrder
FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);
```

Implementing an explicit foreign key has a lot of advantages :

First, it expresses clearly and without ambiguity the information of the logical foreign key. This information can easily be extracted from the DDL code of the database. Moreover, the DBMS will automatically prevent inconsistencies and guarantee the logical foreign key preservation.

2. Implicit foreign key :

   (a) Using other DBMS integrated mechanisms that will simulate a classic Foreign key :

   For example, a set of triggers that will check for every modification of the two tables if the constraint is respected (when a value is added, deleted or updated).

   This method, unlike the previous one, allows a developer to define by himself the order in which data should be inserted in the database. Indeed, if the application code adds data in the order that a classic foreign key would not tolerate and if the developer does not want to modify this code, it could be interesting to manage the foreign key with triggers instead of the dedicated foreign key mechanism.
   On the other hand, mistakes in the triggers code could be done and it would result in serious inconsistency issues in the database. Moreover, if the foreign key is not documented it is likely that it will be lost with time.

   (b) On the application level :

   In this case, the foreign key will not be expressed in any way in the database code but instead, the constraint will be checked through functions in the code of the applications using the database.

   Like the previous method, the main advantage is the freedom of the application code, not dependent on the database for data insertion (or deletion, ...).
   On the other hand, an annoying issue with expressing the foreign key on the application code is that every application related to such database should code its own mechanisms to express that foreign key. Another huge drawback is the fact that it will be very difficult for a database developer to know the existence of the foreign key if it has not been strongly documented. Finally, mistakes can also easily be made.

(c) Expressing it only on the data:

In this case, the constraint is neither declared in the DDL code, simulated by triggers, nor checked by the applications code. Here, only the data can express the implicit foreign key. For example, if all the values of a column are present in the column of another table, it is likely that there is an implicit foreign key between them. But the constraint could also be expressed by a subset of these similar data. Indeed the constraint has not been necessarily completely respected to effectively be an implicit foreign key. For these reasons, locating the implicit foreign keys is not easy and so we are using the work of *Gobert and Maes* [25] for this part (as mentioned in section 2.4).

The main advantage is it requires no computational time for the database and as relational DBMS are beginning to show their limits when faced to large load balancing web applications visited by an important amount of clients, this method of expressing a foreign key could be interesting.
Of course it leads to a huge risk of losing the information of the existence of the constraint through time. Moreover, as the observance of the constraint is never verified, inconsistency in the data is easily brought.

With this classification it is easier to understand that an implicit foreign key is a relational constraint not implemented in the database by the DBMS dedicated mechanism, but is nevertheless present in the logical schema that the developers have in mind.

3.3 implicit Foreign Key issues

Although implicit foreign keys are very common - especially in large legacy systems as explained in section 1.1 - these implicit constraints may lead to relating problems that will be explained in this section.

Once again, instead of listing all issues extensively, the list below will only describe the most impacting of them. The purpose of this section is indeed mostly to raise the reader's awareness of the problematic of the implicit foreign key.

1. The main problem of implicit foreign keys - already raised in chapter 1 - is the high risk of loosing the information of the existence of them. As seen before, generally in legacy systems these implicit foreign keys are not documented at all and with time the people aware of their existences are no longer present or have simply forgotten them.

2. As a result, the initial logical schema can be violated. For example, if a new application is developed on the database and if the developers are not aware of the existence of some implicit constraints, it is likely that the application will not respect the logical schema. So it is also likely that on the same database, some applications are consistent with the logical schema and other are not.

3. This lack of consistency between applications could easily lead to inconsistencies in the database itself. Once again, this could lead to major malfunctions of applications that cannot tolerate this inconsistency.

3.4 Foreign key enforcement

In order to avoid all these concerning issues implicit relational constraints can generate, a process to transform these implicit foreign keys into explicit ones is needed. This process is called **foreign key enforcement**.

With the aim of being as general as possible, we will also consider the adding of a foreign key that was not previously present on the logical schema as foreign key enforcement. we argue that this will allow us to provide a more flexible solution later on this thesis.

So, enforcing foreign keys consists in implementing new foreign keys into the database. This will require database transformations presented in section 4.2 and will cause problems at the application level.

3.5 Impacts of foreign keys enforcement at the applications level

The transformation of the schema is not the only facing problem when trying to add foreign keys. Such manipulation can also have a great impact on the application communicating with the database. This section presents the application level repercussions of such a transformation.

3.5.1 Foreign keys adding Impact

Now, we will develop all cases where the foreign key adding corrupts the good execution of the application code. We will develop all the cases on the example of a delivery application (like Amazon). To visualize these cases we will use as an example the adding of an explicate foreign key between a table “Order” and a table “Client”. Order has an “IDClient” column that references an “ID” from “Order”. Figure 3.3 shows this Transformation.

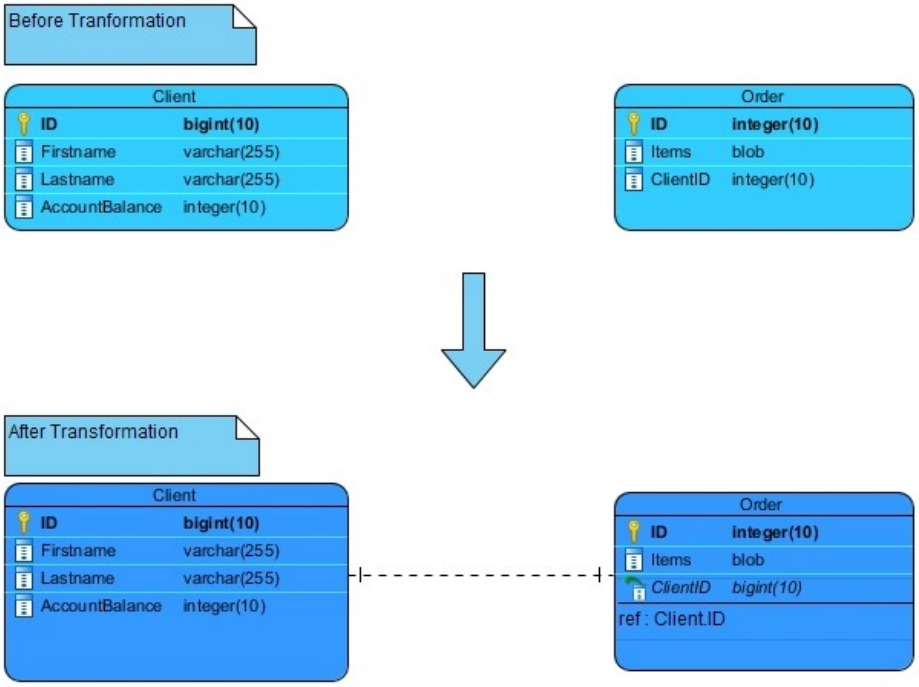


Figure 3.3: Simple NTT transformation

This schema represents a “Numeric Type Transformation” (NTT) defined on the Transformation typology part (see 4.2.1). To perform the adding of reference constraint between “Order.IDClient” and "Client.ID" we need to proceed in two steps. First, change the type of “Order.IDClient” to BIGINT to respect the referenced type matching and then add the foreign key constraint in the DBMS (reasons and steps of all transformations will be developed in the Design part see Chapter 4). To show the greatest impact that such transformations can have on an application, the addition of the foreign key is done in NO ACTION (for deletion, update ...)

To model the application we will use the DAO design pattern (Data Access Object) which is one of the most used for applications that use database to store persistent records.

Figure 3.4 represents the implementation of this design pattern based on our specific example.

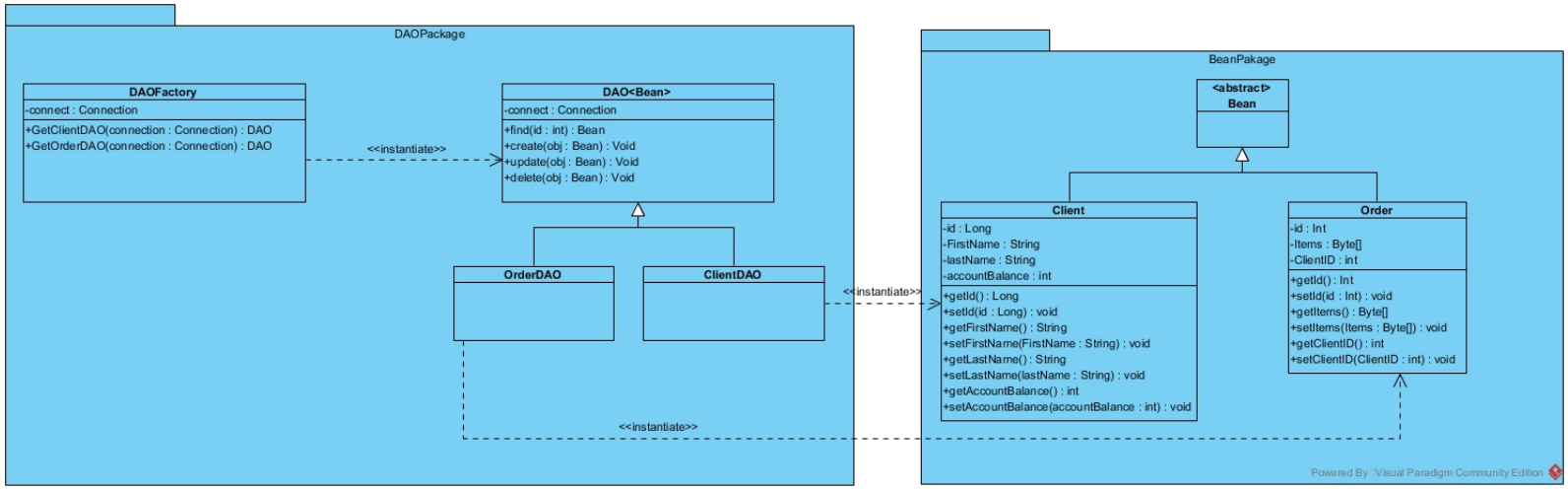


Figure 3.4: DAO design pattern class Diagram [UML]

This design pattern splits the data manipulation into two Objects. The first object is the Beans object. It is a class that stores and structures a database data (typically a DB table line). The second kind of object, the DAO object, takes in charge all direct communications with the database (Select data from tables and generate beans, update/delete/add database line ...)

Operation impact : Adding/update Data

Here, we will see the impact that the adding of the previous foreign key could have on the data adding operation. For that we will use the following use case : "A client registration". For some business reasons, a client is only registered on the database when he places his first order. The following sequence diagram represents the application flow before the Foreign key adding.

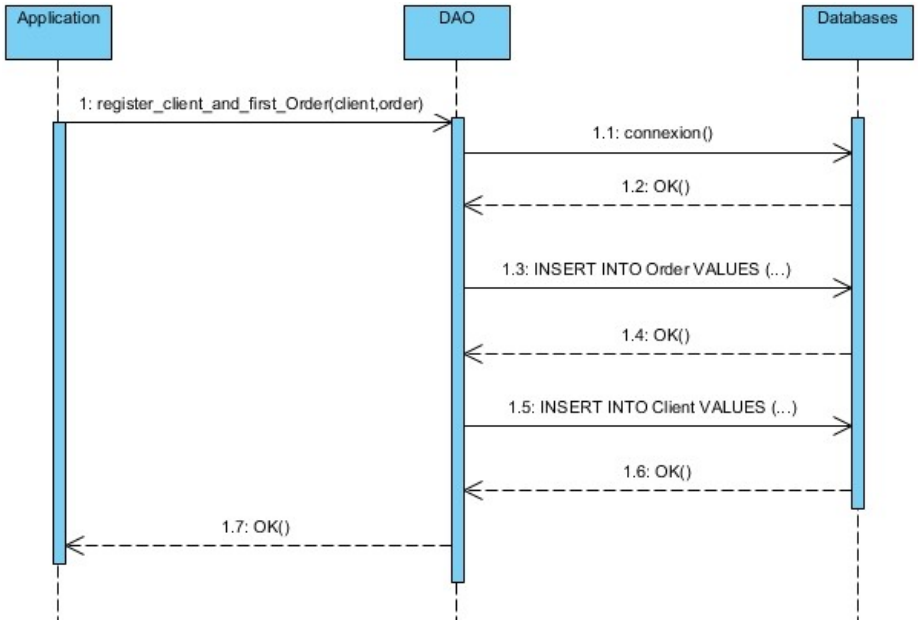


Figure 3.5: Sequence diagram of Data Adding success [UML]

On figure 3.5 we can see that with the "Client" registration operation, the DAO can execute its requests in any sequence order. So, a developer can freely choose to add the first "Order" and then adding the "Client".

Now we will see the impacts of the foreign key on the delivery application:

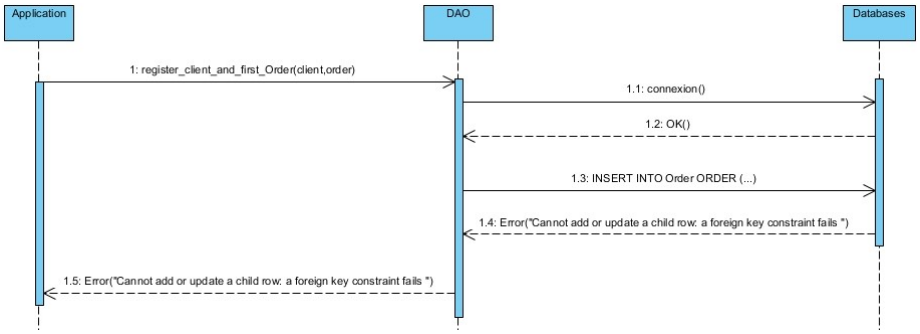


Figure 3.6: Sequence diagram of Data Adding failure [UML]

The foreign key adding have generated new constraints on the execution sequence of the requests. The Client has to be created first because now the DBMS checks at every adding/update that the new OrderID value from Order has as existing reference on Client.

Operation impact : Delete/update data

This case is close to the previous. Here, the use case is : "Delete a client". To keep some data integrity, we will delete all Order that references this client at the same time. But before the foreign key adding it is completely possible to conserve the linked Order. Figure 3.7 shows this use case :

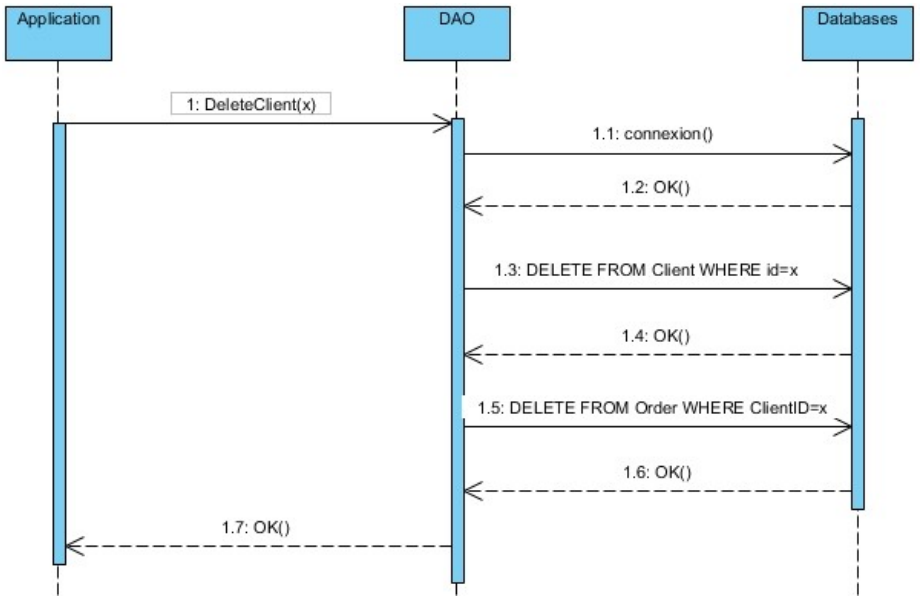


Figure 3.7: Sequence diagram of Data Delete success [UML]

We can see again that there isn't any sequence constraint on the requests without foreign key constraint. Therefore, this implementation is completely correct. But unfortunately when the constraint is added, this implementation is no longer accepted as shown in the figure 3.8:

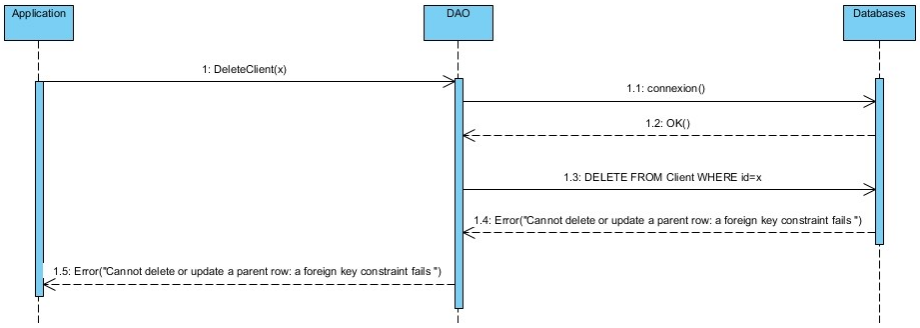


Figure 3.8: Sequence diagram of Data Delete failure [UML]

This scenario will return an error from the DBMS because the DAO tries to delete first a "Client" referenced by at least one "Order". The reference constraints adding on the DBMS has generated new constraints on the sequence request order of the application. Thus, to make possible the deleting we have several alternatives: delete all orders linked to the specific Client, update all Linked Order to a default user or set ClientID to null if it is possible.

Operation impact : Select data

Now, we will see the impact that a database transformation can have on a "select" query. We will visualize this impact via the following example : the application will try to recover all the information of a given "Order" from the DB via a "DAO" object, then we will create a corresponding "Bean" object. This operation is modeled by the sequence diagram shown in Figure 3.9

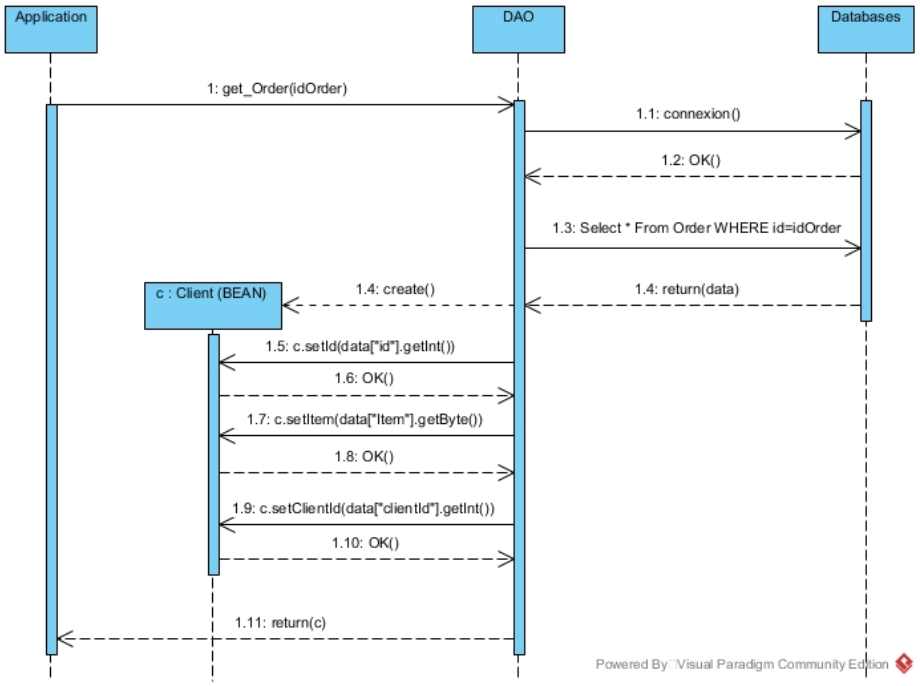


Figure 3.9: Sequence diagram of Data Select success [UML]

However, the addition of the foreign key has the impact of modifying the type of "Order.ClientID" from integer to bigint. This change was necessary to allow correspondence between referencing column and referenced column. Unfortunately, this change, which can be insignificant at the database level, can be problematic at the application level, because the software object that is supposed to represent the object stored in the database must be modified to accommodate the new type for preventing the risk of generating potential errors or exceptions. We can see this case in the following Figure 3.10.

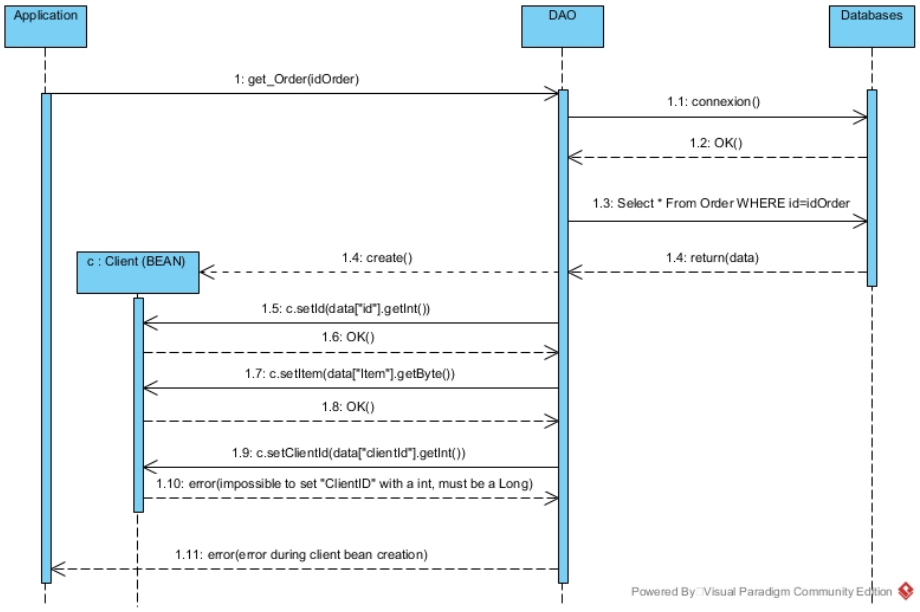


Figure 3.10: Sequence diagram of Data Select failure [UML]

To generalize, a database transformation that adds a foreign key can potentially incorporate a column type change. This type change may, but not always - like in the case of a change into a less global and therefore encapsulate type, require an application level refactor to be able to accept this new type. This refactor operation can be very localized and restricted if the architecture allows it - if it is based on the DAO design pattern and on the BEAN, for example - or global if the database processing operations are not centralized.

This chapter describes the problems that motivate the idea of a process for enforcing foreign keys. In the next chapter we explain the methodology we apply for elaborating that process.

Chapter 4

Methodology

This chapter develops the methodology used for the elaboration of a process for enforcing foreign keys. We first explain which level of automation we chose to apply to the process.

Then, the typology aiming to design an algorithm for transforming the database schema into a new schema where the old implicit foreign keys are explicit is introduced.

After that we describe the inconsistency an explicit foreign key can bring and how to use it through dynamic analysis.

Next, we explain how the impacts that these transformations can generate on the applications based on this database can be analyzed through static analysis.

Finally, we define some metrics useful for the solution.

4.1 Levels of automation of the process

In the previous chapter, we describe the issues with implicit foreign keys and conclude to the necessity of a process making them explicit.

We defined three different levels of automation that could be applied to such a process :

1. Automatic
2. Manual
3. Semi-automatic

All these levels have inherent qualities and flaws.

With the automatic level, the process would have an integrated algorithm that would decide what to do with each foreign key - that is to say what database transformations would be needed for the adding of the foreign key and what transformations in the applications code should be made.

The main advantage of this level is obviously that the user has nothing to do; the process is fully automated and will take every decisions for her/him. But that particular benefit is also its biggest flaw : the user has no control on the decision making process and it has been shown in the previous chapter that some database managers do not tolerate that.

Moreover, if such an algorithm is fully conceivable for the database transformation part, it is however extremely complex to automatically determine which part of an application code should be modified. This particular point will be developed later in this thesis.

With the manual level of automation, the process would go through each implicit foreign key one by one and, for each of them, would ask the user what to do : add it explicitly, abort this foreign key (keep it implicit) or others (like simulating it with triggers for example).

If the adding option is chosen and if that requires database transformations, then the solution will ask the user what it should transform and in what.

This time, the user has complete control on the decisions made but on the other hand the process is quite limited and its unique contribution is that it will apply the transformations decided by the user.

Very soon in the development of our thinking it became an evidence that an automatic database transformation process will not often be used in legacy systems (The OSCAR case described later supports this assumption). The reasons for this are many and various. Here are listed only some of them; this particular point being more deeply developed in chapter 6.

1. Database managers of these systems will not dare to use a process that will modify the database without their strict consent.

2. Database managers will probably prefer to make explicit only a part of all the implicit foreign keys : obviously the less costly ones first.

3. As stated in the introduction, the database of the legacy system may require a migration that cannot be done in one time but have to be iterated. Thus, all the implicit foreign keys cannot be automatically treated.

The manual level of automation was also very quickly dismissed because of its lack of user assistance.

With the semi-automatic level, the process will act as a decision helper tool. Indeed, for each implicit foreign key, it will ask the user what to do but at the same time, it will provide him with helpful information that comes from its integrated algorithm (similar to the automatic level) that will allow him to make better decisions.

This level seems to be a satisfactory balance between control on the process and its contribution importance. The automatic level is too dangerous for the database managers because it could lead to unwanted changes and the manual level does not help him enough for making the decisions.

As we have seen before, adding a foreign key requires two separate manipulations :

1. A database transformation manipulation

2. An application source code changes manipulation

It is conceivable to develop an algorithm that will determinate the best database transformations that are needed and apply these changes (or the ones decided by

the user if he does not agree) but it seems unlikely to develop a module for the process that would apply changes directly in the source code of applications. For that last part, it will only inform the user where, in the source code, errors could rise because of the new foreign key, leaving to the user the task to effectively deal with these potential issues.

4.2 Database transformation

In order to facilitate the design of a database transformation algorithm, it is important to first formally define what a database transformation is.

A database transformation can be defined by a function $DT : DS \rightarrow DS$ where DT is a Database transformation and DS is a Database schema.

DT can be decomposed into a sequence of atomic database transformation (ADT) operations: (t_1, t_2, ... , t_n)
where DT(ds_1) = ($t_n \circ ... \circ t_2 \circ t_1$) (ds_1) = ds $_2$ $\wedge ds_i \in DS \wedge t_j \in DT$

These atoms could be:

1. A foreign key adding
2. A column type modification (that includes column value conversion)
3. A line suppression

4.2.1 Transformation typology

To facilitate the comprehension and the process of the transformations, we sorted them into 3 main transformations and other sub transformations :

1. Database Transformation
   (a) Matching with Basic Transformation (MBT)
   (b) Matching with Values Mismatching Transformation (MVMT)
   (c) Length Mismatch Type Transformation (LMTT)
   (d) Type transformation
      i. Numeric Types Transformation (NTT)
      ii. AlphaNumeric Types Transformation (ANTT)
      iii. Time Types Transformation (TTT)
      iv. Different Type Transformation (DTT)
2. Empty Transformation
3. Impossible Transformation

4.2.2 Database Transformation

Matching with Basic Transformation (MBT)

This case happens when there is a perfect matching - same type and same length - between the source column of the foreign key and its destination.

example:
try to add foreign key between : Table1.c1 -> Table2.c2 on the DB diagram from Figure 4.1

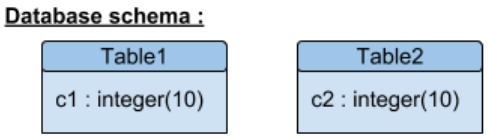


Figure 4.1: MBT example

Matching with Values Mismatching Transformation (MVMT)

This case happens when there is a perfect matching between the source column of the foreign key and its destination but when some values mismatch.That means that it is not possible to satisfy the referential constraint, in state, because some values present in the database do not respect it. New values must either be added to the reference table. Either, delete the values that have no referents.

example:
try to add foreign key between : Table2.c2 -> Table1.c1 on the DB diagram from Figure 4.2

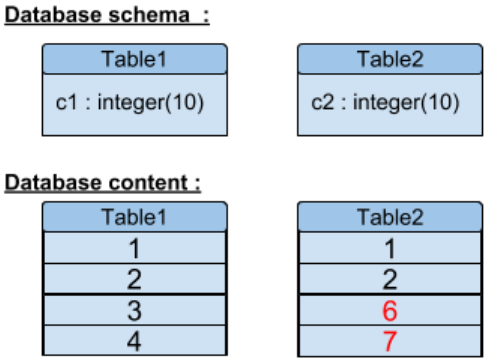


Figure 4.2: MVMT example

Length Mismatch Type Transformation (LMTT)

This case happens when the types of the source column and the destination column of a foreign key match but the length of these types mismatch.

example:
try to add foreign key between : Table2.c2 -> Table1.c1 on the DB diagram from Figure 4.3

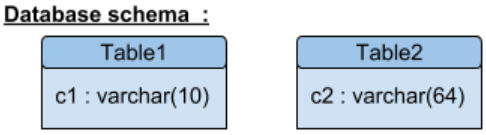


Figure 4.3: LMTT example

Type transformation

This case happens when there is a type mismatching but the types are both in the same meaningful set.

example:
try to add foreign key between : Table2.c2 -> Table1.c1 on the DB diagrams from Figure 4.4

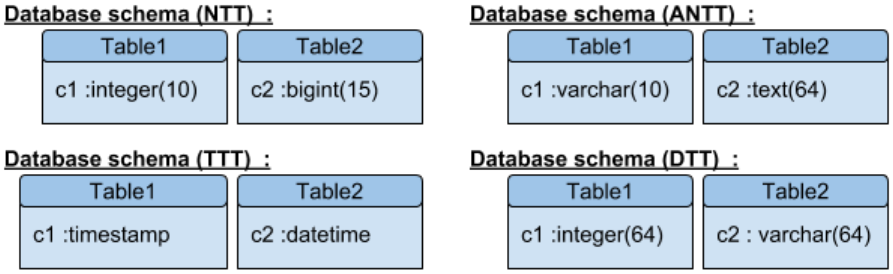


Figure 4.4: Type transformation example

4.2.3 Empty transformation

Specific case when we do not have to proceed any transformation. This case happens when the foreign key is already implemented.

4.2.4 Impossible Transformation

Specific case that happens when we don't find any possible transformation that can allow the adding of the foreign key. That can mean that the foreign key is wrong or that we can't proceed a safe automatic database transformation.

We can already list three cases where a transformation would be impossible:

1. If one of the tables concerned by the foreign key is not present in the database. As we work with systems in perpetual maintenance, the database tables could be deleted or renamed over time. So, if we find or compute a foreign key of an X version of the database, we will not always know how to add it in an X+1 version.

2. If we're between two completely incompatible types. In this case, it would not be possible to convert the values present in a column to add the referential constraint. For example, try to add a foreign key between a column typed by a BLOB and a DATETIME. We also add here the case where we try to add a foreign key between a signed type and unsigned because here it is impossible for us to determine the one of the 2 types is the most appropriate one

3. This last case is the "other" case, it would be all the cases where our system is not capable of performing the transformation, but does not know why.

We now have the basic to design a transformation algorithm that will help a user in his decisions. That particular algorithm will be elaborated in next chapter.

4.3 Program adaptation

This section will explore two ways of bringing information to the user about the impacts the new foreign keys can have on the applications that use the targeted database:

1. Dynamic analysis : first, a discussion about the tolerance of inconsistencies in a database is required to understand how emerged the idea of the "trigger logger" that will be presented afterwards.

2. Static analysis : based on the work of *C. Nagy, L. Meurice, and A. Cleve* [26]

4.3.1 Dynamic analysis

Tolerating Inconsistencies

When facing inconsistency in a database, most people would think it should be resolved immediately, considering it entirely as an issue. But when performing a database migration, the inconsistencies are inevitable. Thus, solutions to deal with them are required. However, "dealing with them" does not necessarily mean resolving them directly, because in large legacy systems it is an impossible task to achieve. Indeed, in these cases, the usual upgrading procedure is to process the migration iteratively because it is nearly inconceivable to stop the system entirely to do the modifications. This policy of upgrading parts after parts will obviously engender some more inconsistencies because some parts of the systems will be of the new version while other parts will still be of the old version.

It seems then that there are few options other than tolerating - hopefully only temporarily - some of the inconsistency.

In this thesis it is primordial to take into consideration the inconsistency brought by the addition of new foreign keys. If the option of banning it entirely is chosen, it is most likely that a lot of implicit relational constraints could not effectively be safely added without imperatively changing the application code first.

A decent solution to this problem, according to *Balzer*[7], is to "weaken the constraint to authorize the exceptions". That way, the applications still work and, at the same time, even if the constraint is not implemented, it is at least documented and notified so that the developers will hopefully implement it eventually.

It is important to understand that the inconsistency can be treated as crucial information about the foreign key and the next section will described a treatment of this information.

Trigger logger

This section describes a simple methodology we called "trigger logger" that allows to weaken the constraint of a foreign key by tolerating inconsistency and at the same time collecting information about this inconsistency. The idea is to create a new table in the database that will collect the information gathered about any violation of the new foreign keys whereas the database itself still accepts these violations. To do that, instead of declaring plainly an implicit foreign key, triggers will be declared in the source and destination table. These triggers purpose is to let any violation of the foreign key pass but at the same time to log these violations into the new created table.

As the new table is a "log table" and the basis of this mechanism is triggers, the name "trigger logger" seemed appropriate.

Figure 4.5 shows a simple and schematic view of the "trigger logger" methodology.

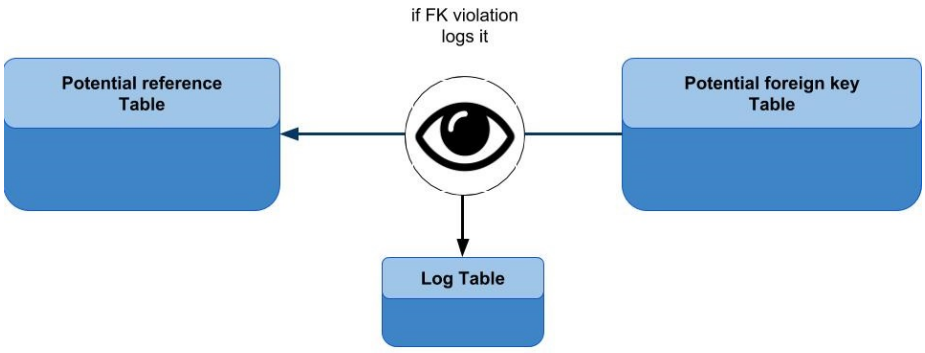


Figure 4.5: Trigger Logger methodology

The details of the design and the implementation will be developed later in this thesis but at this point it is important to understand that this mechanism - even if its main purpose is to allow the developers of a database not to implement all the foreign keys in one iteration of a migration - will here essentially be used as a mean of estimating the potential risk of adding formally a given implicit foreign key. This point will be developed in the last section of this chapter.

Finally, let us draw the attention of the reader to the importance of understanding plainly that the "trigger logger" mechanism can only return information about the violations on a temporarily period of time. Thus, if no violation of an implicit foreign key arises on a given period, it is impossible to affirm firmly that making that particular key explicit will not cause any issue in a undetermined period.

Nevertheless it is still very interesting to have that estimation of the probable frequency of violation of a given implicit key.

4.3.2 Static Analysis

The static analysis of a source code to detect where erroneous queries are made in the code has already been developed by *C. Nagy, L. Meurice, and A. Cleve*[26]. They developed a tool called DAHLIA for Database ScHema EvoLutIon Analysis. If the reader is interested in the functioning of this tool, a brief explanation had been started in the "state of the art" chapter of this thesis. Here, we will simply say that this tool can allow us to extract useful information for the user of our process about the locations in the code that should be modified to accept a new foreign key.

Unfortunately we had not enough time to integrate plainly this tool into ours. Nevertheless, a short explanation of the mechanism we wanted to develop for retrieving these queries will be given in subsection 5.2.2

4.4 Decision helper process

Now, we dispose of enough information to help the user take good decisions. The decision helper process should, for each foreign key notice the user of these information in a very simple way so that the user can decide quickly (if there are hundreds candidate foreign keys to be treated, he cannot permit himself too much time on each one of them). A quick and clear way of giving him these information would be to condense them into representative metrics that, combined, would gave the user a faithful representation of the situation.

The next section will explain which metrics we chose to use in the process, what they represent and how they can be calculated.

4.4.1 Explicit Foreign key metrics

We choose to express three different metrics for the decision helper process :

1. The importance of the database transformation that could be in order to add the candidate key which we called TRANSFORMATION MAGNITUDE :

(a) If the type of a column has to be changed.

(b) If there are value mismatching between the foreign key column and the destination column.

(c) If there are cascade transformation required.

For the whole transformation typology, see subsection 4.2.1.

2. The probability of errors that would occur after the adding which we called VIOLATION FREQUENCY.

3. The importance of the modification on the application level that would need to be undertaken in order to safely add the foreign key which we called APPLICATION IMPACT.

In subsection 4.2.1, we saw that it is possible to categorize each transformation of the database. Giving for each transformation an arbitrary cost is not complicated. Thus, it would be quite easy to compute the "TRANSFORMATION MAGNITUDE" value through our database transformation algorithm.

The "VIOLATION FREQUENCY" value could be given by the results of the dynamic analysis : analyzing the log table of the trigger loggers should be sufficient.

The "APPLICATION IMPACT" value is more difficult to calculate but we are confident that it is achievable through the static analysis we described.

If we could give a value for each of these three metrics it would be possible to calculate, for every foreign key, a value corresponding to the global cost of the addition of that foreign key and then it would be very easy to determine which foreign keys are the safest for making them explicit without overextending cost.

These new informations should help a user by providing him a quick and global view of the potential dangers of any candidate foreign key and also, with them, a user will be able to circle down a smaller number of foreign keys to treat.

Nevertheless, a scale has to be arbitrary attributed for each of these metrics that could rapidly tell the user if the value for a metric is good or not (with a color code for example - see figure 4.6.

We have not yet addressed this particular point but we suggest a mechanism that would allow the user himself to determine these scale so that they correspond exactly to what he is expecting.

4.4.2 Requirements

There are still several problematics that need resolutions.

The domain experts

Often in legacy systems - the case study described in chapter 7 being an example - the developers, experts in the domain of database, are not the people allowed for actually modifying it themselves. The persons in charge, the real database managers, are only asking the developers advices on what to do but they have

the final words on the actions undertaken. This decision helper process targets being the experts and not the managers, a workaround had to be found.

A good alternative is designing the process for sending a script that will fetch all the needed information, return them back and then display all the treated input that are useful for the experts about every foreign key. This mechanism is explained in chapter 5.

The data privacy issues

There is a last problematic that has to be addressed that this time does not concern implicit constraints but the data itself from the database. It is not rare that large legacy systems contain certain data that are under legal legislation like medical data as it will be shown in chapter 7. But privacy legislations does not only include medical data but also bank data, or, in general, any personal data.

As these private data represent a non-negligible amount of data, their cases should also be taken into account. But what are the issues that a process for making implicit foreign keys explicit has in the context of private data?

1. In some legacy systems, managers of databases containing private data will not dare to use an externally developed process. They will rather either use a secure and internally developed tool or execute all the manipulations by themselves.

2. These database managers could not be experts in the database field - like we will see in the case study in chapter 7 - and so, if decision has to be made, an external expert team should be consulted. In that situation, any information the process could bring to that external team in order to help them propose decisions cannot contain any private data.

3. Last but not least, these database managers will always want to give their consent for every modification of the database. That particular point excludes the possibility of a fully automated process as already discussed in section 4.1

It is obvious that if we want to transform implicit foreign keys into explicit ones it will be essential to propose solutions for the case of working with private data where the process should never manipulate that kind of data. These solutions will be developed later in this thesis.

Figure 4.6 gives a visual summary of all the mechanisms that are used for the improved decision helper process.

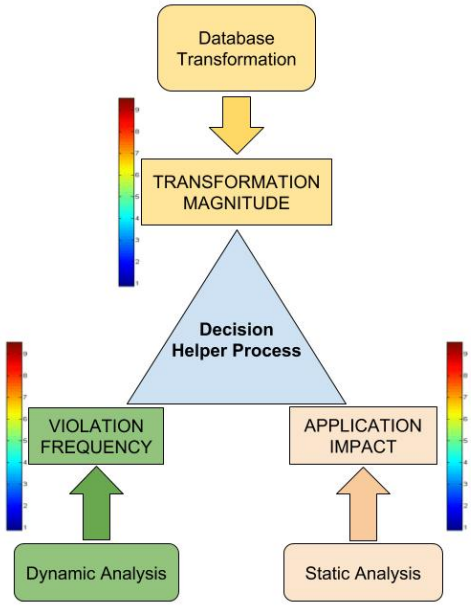


Figure 4.6: The Decision Helper Process

Chapter 5

Design

This chapter is dedicated to the design and the algorithms that are to be establish and developed. This purpose of this part is to think about the way of creating a generic solution that can help database managers to correct their unimplemented foreign keys.

5.1 Database Transformation

To be able to successfully implement an efficient and maintainable software solution, it is imperative to subdivide our main problem into sub-problems and then solve them. To do this, we need to develop a set of tools that we will combine to get our final solution for computing the database transformations that are required for the adding of candidate foreign keys. This section will focus on the design and utility of these tools.

The tools that will be discussed in this part are the following: first, a tool used for the direct manipulation of a database (reading data, modifying schemas, modifying data, etc.), then, we will talk about a tool in charge of analyzing the context of the database and according to it, propose the most adapted transformation, and finally, a remote diagnostic tool to avoid the maintenance problem of databases storing confidential or sensitive data will be presented.

This section will focus on the design and utility of these tools, the specific implementation aspect will be discussed in the next chapter. (see chapter 6)

5.1.1 EasySQL : Abstract Database manipulation

To facilitate the database schema transformation it is important to have an SQL abstraction. With it, we can easily encapsulate a lot of database manipulations (select, update, alteration, add/delete data, ...) on some code classes. This abstraction makes the software code clearer, more readable and manipulable.

This tool could also be very useful to extract meta-data from the database schema. It can automatically create "Table" software object class with all information taken from the database schema such as the primary key, foreign keys list, column name, type, charset, default values. The SQL manipulations

encapsulation is based on the "abstract factory" design pattern that is used to facilitate the request creation and to improve the potential interoperability with multiple DBMS.

A SQL query do/undo feature was added to make possible the rollback of the database in the previous state if there is any problem or if the transformation has too many impacts on the application that uses the database.

Figure 5.1 represents the class diagram of EasySQL, the implementation of this tool that we have developed (see 6.1.1).

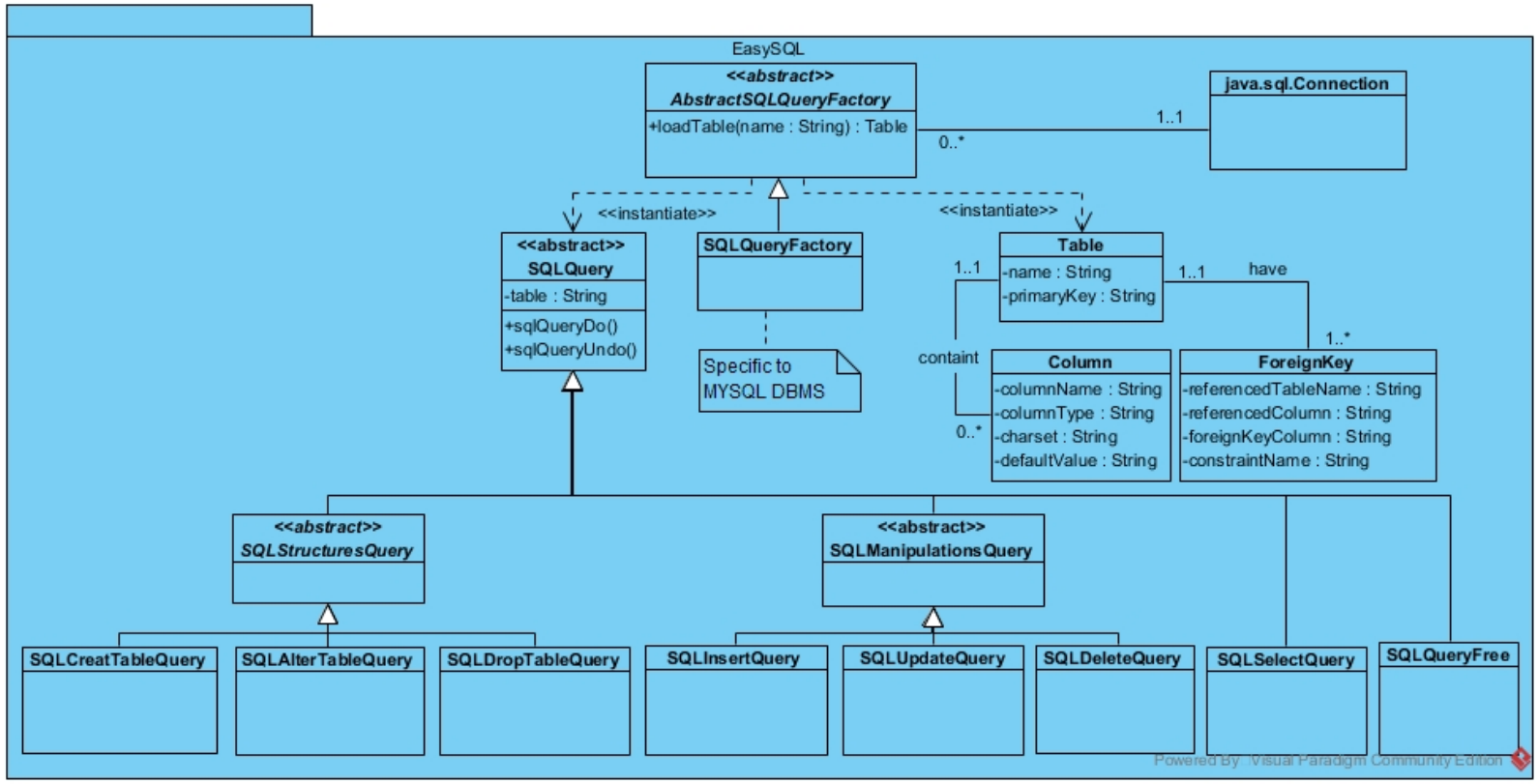


Figure 5.1: EasySQL class diagram [UML]

5.1.2 Context Analyzer

The context analyzer is the real heart of this research. This application takes as parameters a given database and a list of potential foreign keys (given by another research project by *M. Gobert and J. Maes*[25]).

The solution uses our EasySQL library to extract meta-data from the given database and then computes the best transformation that should be processed in order to add a given candidate foreign key. This transformation should be the best choice to conserve the data integrity. But this choice is only based on the database meta-data and another choice could be a better image of the initial schema logic.

For example if a foreign key have this structure :
clientNumber : int $\rightarrow$ *CommandeClientNumber : BigInt*
and the Bigint type is not needed, it is just a maintenance mistake. The application will foster the transformation of ClientNumber into BigInt to be sure to store all potential data but it is not the most optimized choice in this particular case.
It is one of the many cases that demonstrate why this tool is not full-automatic but is a decision support tool.

So, the best transformation choice uses the transformation typology defined earlier (see 4.2.1 Transformation typology). The following Figure 5.2 shows the algorithm to choose which type of transformation is associated with a particular foreign key.

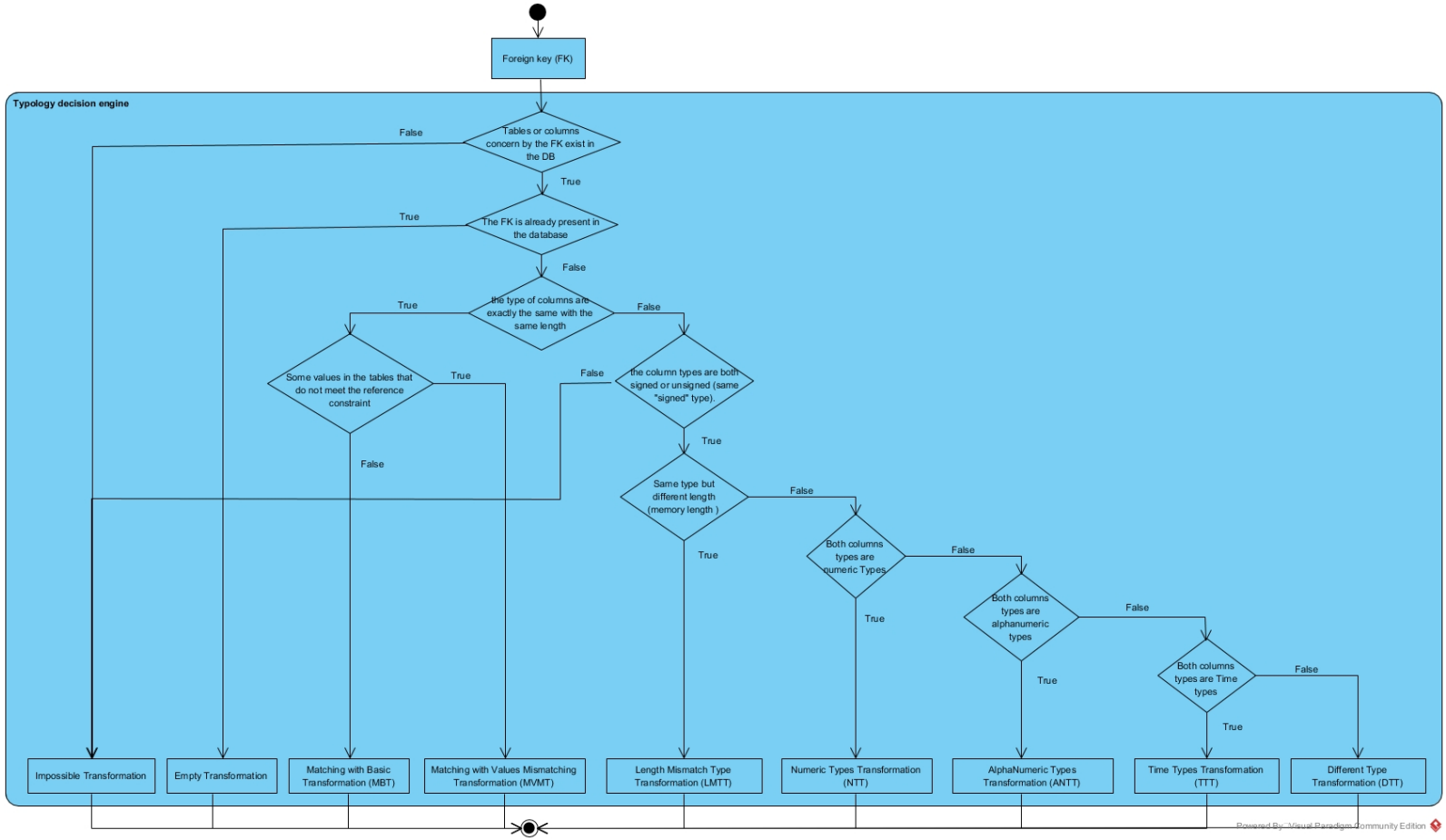


Figure 5.2: Typology decision engine [UML]

Now, let us talk about the different steps that must be performed to allow the concrete addition of a foreign key in a database. To do this, we will detail the different strategies to be established depending on the types of transformations seen and determined above.

Matching with Basic Transformation (MBT)

Proposes to add the foreign key without processing any other transformation.

Matching with Values Mismatching Transformation (MVMT)

After the detection of this case, the "Context Analyzer" leaves 3 different choices to the user : delete on cascade the unmatching values, set them null or abort the fk adding.

Another possibility would be to find and execute operations that could potentially resolve these mismatching values. These operations could be switching the unmatched values with the more potential matched values according to some statistical algorithms, performing upper case, lower case or more complex manipulations.

We choose not to develop this last possibility because it is very difficult to find generic data conversion and it could be dangerous for the database integrity.

Finally, it is worth noting that some DBMS, like Mysql, include built-in data type conversion functionalities. Effectively, in a lot of cases, the data will be converted with no problem at all (for example, converting a varchar column containing numbers - "1", "2", "235", ... - into an INT will not cause any trouble in mysql).

Length Mismatch Type Transformation (LMTT)

Proposes to the user to transform the less precise type into the most precise before adding the foreign key.

Additional warning:
This transformation (from less to more specific) can cause problems with some types (like decimal types) because the type parameter has some constraint (ex: float(m,d) -> (m > d) and (m+d have a maximum length))

Type transformation

To perform this transformation we have to list which types can represent the similar piece of information and sort them from the less "large" to the most "large". And so the context analyzer will propose to transform the less "large" type into the most "large". This type hierarchy is presented in the Table 5.1

Empty transformation

The context analyzer just informs the user that the foreign key is implemented and goes to the next candidate.

mysql example :

Set	Classification
INT type (NTT)	TINYINT <SMALLINT <INT <MEDIUMINT <BIGINT
Alpha numeric type,(ANTT)	ENUM <CHAR <VARCHAR <BLOB <TEXT <TINYBLOB <TINYTEXT <MEDIUMBLOB <MEDIUMTEXT <LONGBLOB <LONGTEXT
Time type (TTT)	TIMESTAMP <DATETIME
Exception	• We excluded TIME,YEAR,DATE of Time type because their transformations don't have a lot of meaning to a database transformation with the aim of adding foreign keys • We didn't take into account the transformations between FLOAT,DOUBLE,DECIMAL because they are different internal representations that can make impossible good values transformations. • FLOAT is an approached value with single-precision encoding • DOUBLE is an approached value with double-precision numbers encoding • DECIMAL is an exact numerical value

Table 5.1: mysql type typology

Impossible Transformation

We cannot analyze this foreign key (inexistent table, column, ...), therefore we simply notify the user of the impossibility to treat this case then goes to the next candidate.

The following figure 5.3 is a class diagram of the context Analyzer package. We can see on it a partial representation of EasySQL developed earlier and the Transformation package based on the *Command* design pattern that stores the transformation typology developed earlier.

The Figure 5.4 shows an algorithm showing the steps required to allow a foreign key to be added to a database

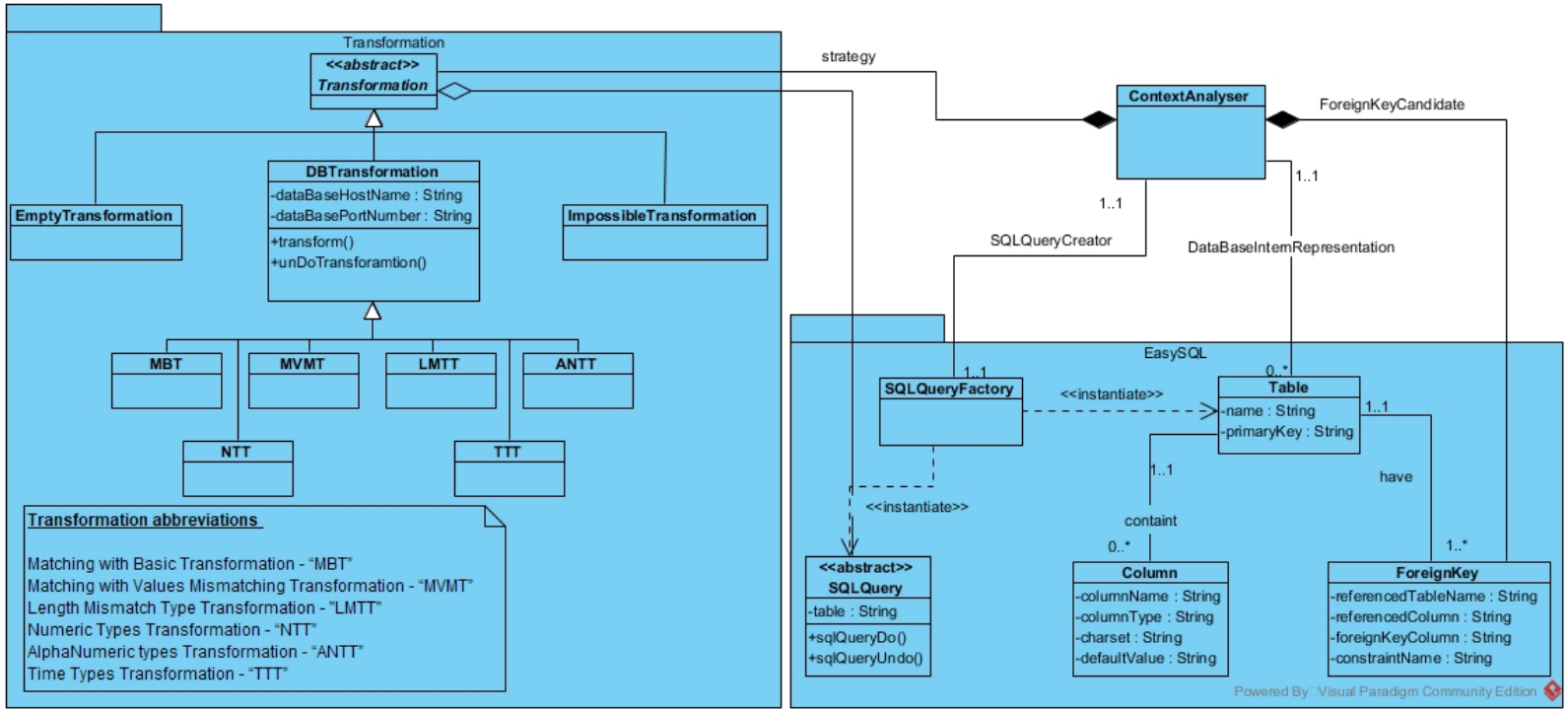


Figure 5.3: Context Analyzer class diagram [UML]

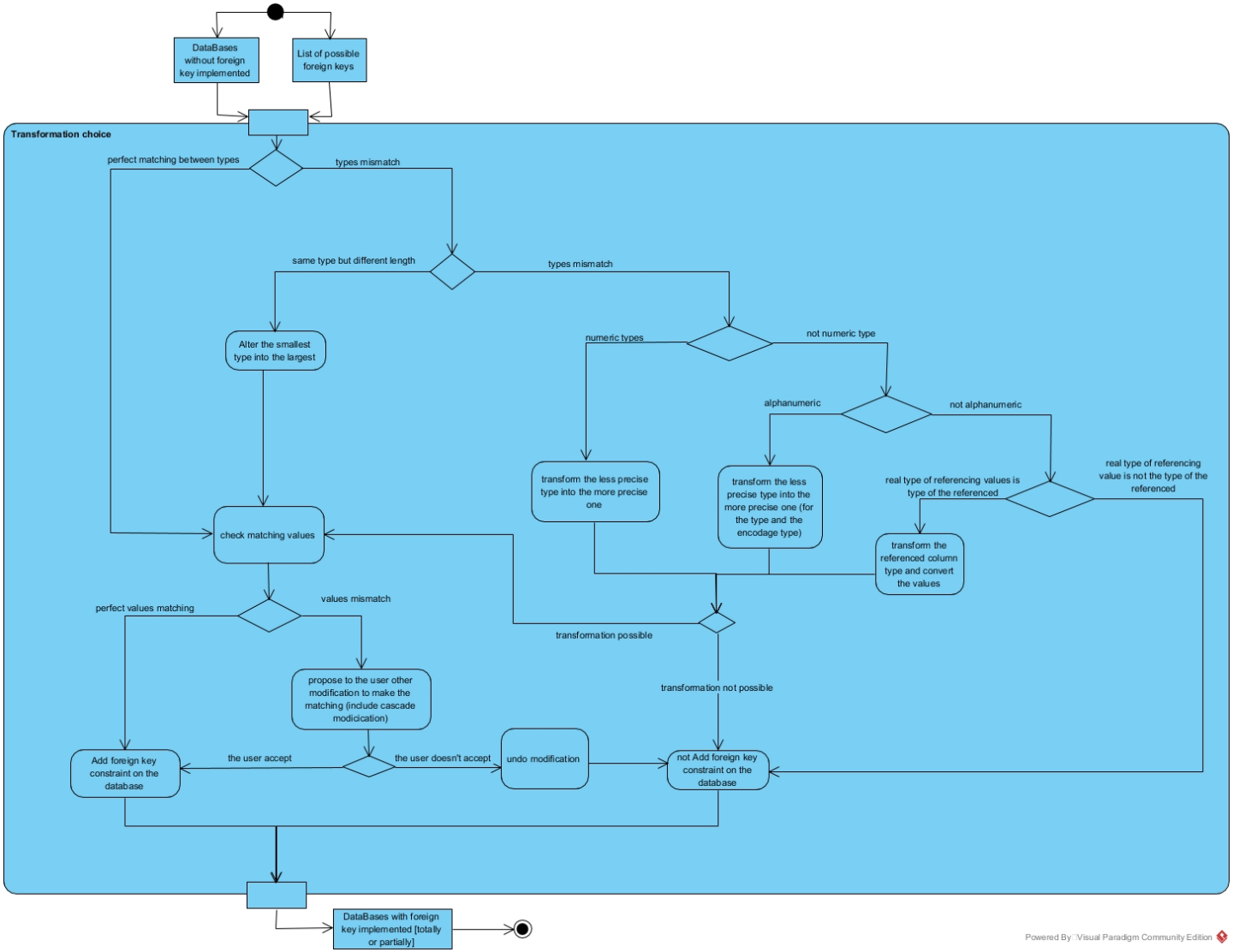


Figure 5.4: Transformation choice algorithm [UML]

5.1.3 Remote Diagnostic

In the first version of the solution we provided, we were in a perspective where the database maintenance and transformation team had full access to the database. However, the reality of the environment is quite different and we quickly realized that for security reasons, for legal constraints on the protection of certain data judged critical or in other cases this access could cause problems.

In response to this, we designed a remote diagnostic system that could be run by someone with the required credentials. This tool, based on the context analysis mentioned above, would take the list of candidate foreign keys and generate a report, structured via a language such as XML or JSON, containing the transformations recommended by our resonance engine (see Figure 5.4). This report would not contain any information considered sensitive such as values being in the database, but would only contain information relating to the impact of the addition of the foreign key like the number of values not respecting the referential constraint, the potential type changes of the columns, the cascade transformations that it implies, ...

Here is the total list of information represented in this report. For each candidate foreign key it contains :

1. The advised new type (the same that the context analyzer will propose)
2. If there is a potential unmatching compatibility between signed or unsigned type (Boolean value)
3. If there is an encoding matching or not (Boolean value)
4. If the context analyzer does not find a way to add the foreign key (Boolean value)
5. The list of potential cascade foreign keys (concerning the candidate foreign key column)
6. The list of potential cascade foreign keys (concerning the candidate referenced column)
7. The advice transformation target (referenced column, foreign key column or both)
8. If the foreign key already exists (Boolean value)
9. The transformation type (MBT,MVMT,LMTT,NTT,ANTT,TTT or DTT)
10. The number of unmaching values
11. A potential message from the context analyzer (Warning or reason why the foreign key adding is impossible)
12. The candidate foreign key information

The report also contains a set of information concerning the database schema, a series of information on the tables concerned by the candidate foreign keys. These tables are the source and destination tables of the candidate referential constraints as well as all tables already having a referential constraint with them (referencing table or referenced table). The total list of stored information about these tables is as follows:

1. the table name
2. the primary key column name
3. the list of the column of the table with the following information :
   (a) the column name
   (b) the column type
   (c) the default value of the column
   (d) the charset of the column
   (e) the value true or false if the column is unsigned or not
4. the list of foreign key concerned by this column with the following information :
   (a) the referenced table name
   (b) the referenced column name
   (c) the foreign key table (the table with the referential constraint)
   (d) the foreign key column (the column with the referential constraint)
   (e) the name of the constraint stored on the database

With this report and the information it contains, we have all the useful data for our decision support tool to interact with a user in order to generate useful output - for example a SQL script taking care of the addition of approved candidate foreign keys. It therefore represents a total substitute for a direct connection to the database.

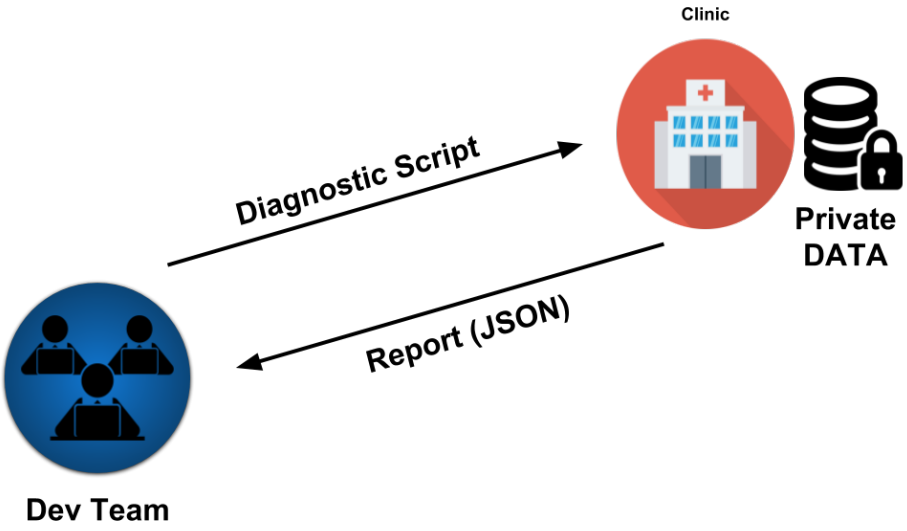


Figure 5.5: Remote diagnostic schema

Figure 5.5 is a very schematic representation of the remote diagnostic tool on a particular case. It is the case of a hospital that contains private data which outsources their database management by a extern development team.

5.2 Program adaptation

5.2.1 Dynamic analysis

For the dynamic analysis part of the tool we developed the "trigger loggers" described in section subsection 4.3.1. The purpose is to retrieve information about violations of simulated constraints, that is to say that it should record the violations but, at the same time, tolerate them.

To that end, a log-table that will record the information is needed. In this log-table several information should be recorded for each violation :

information	Purpose
The Foreign Key name	To quickly identify the critical foreign key
The source table name	To record the source table location of the problematic data
The source column name	To record the source column location of the problematic data
The destination table name	To record the destination table location of the problematic data
The destination column name	To record the destination column location of the problematic data
The type of the violation	To record the problematic operation (insert, delete, ...)
The date and the time	To record the exact time of the violation

Table 5.2: Log-table components

Once that log-table has been added, the triggers strictly speaking can be added. These triggers should check, for each wanted implicit foreign key :

1. Insertions in the source table : if a data is inserted in the source column while this value is not in the destination column.
2. Deletes in the destination table : if a data is deleted from the destination column while it is still in the source column.
3. Updates in the source table : if a data is updated in the source column while this updated value is not in the destination column.
4. Updates in the destination table : if a data is update in the destination column while its previous value is still in the source column.

The construction of the log-table and the triggers will be explained in subsection 6.2.1.

5.2.2 Static analysis

To retrieve information about the potential erroneous queries executed by an application on the database for an implicit foreign key made explicit, as introduced in subsection 4.3.2, our choice was to use a tool called DAHLIA that is developed in the purpose of calculating the location of erroneous queries.

As already said, we unfortunately did not have enough time to think it through. Nevertheless, it is possible to start the thinking and give some clues to how our tool could use DAHLIA to help its user.

DAHLIA is able to analyze statically the source code of an application to rebuild and extract the database queries formed inside this application (even if the built queries have parts in different classes). All these rebuilt queries are stored in a given format file (DAHLIA uses JSON) that can be analyzed.

So, it is technically possible for our tool to launch the query rebuilding process of DAHLIA on a given application, to retrieve the output file and then to analyze it to produce information to the user.

For the moment, we mainly explored the possibility to retrieve the locations of all the queries that could potentially be affected by a given foreign key the user wants to make explicit, but profound research in this particular field should be undertaken in order to be able to produce more detailed information about these queries - which ones will really be an issue when the foreign key will be added, for example.

Since we had very little time to devote ourselves to this part of this thesis and even if we started to implement a short analysis of the output given by DAHLIA, we will not present an implementation for this aspect of the tool in the next chapter.

5.3 Visualization tool

All the previous designed tools are useful but it is essentially the grouping of all of them together that will precisely meet the needs outlined in chapter 3. The global decision helper tool, as said in chapter 4, should provide a visualization of the level of issue or risk a candidate foreign key has. To that end, 3 different metrics were defined and they all can be calculated by these previous tools :

1. TRANSFORMATION MAGNITUDE :

   This value can easily be calculated by the Context Analyzer Tool (subsection 5.1.2) based on the Database Transformation Algorithm (Figure 5.4), or, if the database is not directly accessible, by the Remote Diagnostic Tool (subsection 5.1.3) which is itself based on the Context Analyzer Tool.

   Indeed, as these two tools use the transformation typology defined in subsection 4.2.1, if a weight is attributed for each type of transformation, the tools - because they can calculate all the required database transformations for a given candidate key - could then also calculate all the corresponding weights of these transformations, and finally add them to produce the needed value.

2. VIOLATION FREQUENCY :

   This value can also be easily calculated by the Trigger Logger Tool (subsection 5.2.1).

   This time, the tool just has to calculate the number of violations for a given candidate foreign key to produce the needed value. We attract the user attention to the fact that other values could be calculated if weights are applied to the different types of violation (maybe a user considers an insert violation as more troublesome than an update violations?) but here we will simply consider the first solution.

3. APPLICATION IMPACT :

   There are several ways for the Static Analysis Tool (subsection 5.2.2) to calculate this value :

   (a) Add the number of potential impacted queries for a given candidate foreign key (very simple)

(b) Add the number of the surely known impacted queries for a given candidate foreign key (demands some serious research in static analysis)

(c) Same as the two previous ones but this time giving weight to the queries according to their nesting level in the code (we are confident that DAHLIA could provide this information)

In subsection 4.4.1, we discussed briefly the possibility to let the user decide himself of the scales for each metrics (for deciding when a value is considered acceptable or not). The other possibility is to hard-code them arbitrary during the implementation of the tool which is not a viable solution as different users will have different needs.

Unfortunately, like for the previous section, we had not enough time to concentrate ourselves on this tool other than designing it in general terms so that no implementation will be presented in the next chapter.

Chapter 6

Implementation

In this chapter, we will discuss the implementation of the algorithms discussed in the previous chapter. We will not detail the code in-depth and will focus on the most original aspects developed or requiring further research.

The full code is available on Github at https://github.com/hcarlUnamur/prodacon2. This represents approximately 75 classes and over 9000 lines of code.

6.1 Database transformation

6.1.1 EasySQL

The first part of the solution implemented is EasySQL. It is a SQL query representation at a software object level, so we can see it as an abstraction of an interface with which our software solution interacts to communicate with the database. It facilitates query creation, execution, rollback and metadata recovery from the database. The metadata that can be searched are the following:

- information on tables like :
  - their primary keys
  - their foreign keys
  - column information such as :
    - their names
    - their type
    - their encoding
    - their default values

The way to extract this data will be specified in the following point.

At the structure level, EasySQL is a java implementation of the class diagram presented in the design part (see Figure 5.3). This library is based on the Factory design pattern to allow it to be adapted to any DBMS as well as to facilitate the instantiation of these objects. Our current implementation is only compatible with MySQL DBMS, but a multi-Platform version compatible with

Microsoft SQL server, DB2 or Oracle Database could be considered in the future.

The following sections will detail 2 important features of EasySQL which are database metadata extraction and do/undo functionality.

Meta-data extraction

In order to be able to choose which transformations are necessary to make an implicit foreign key explicit, it is imperative that we extract the information on the current schema structure from the database. This information is called metadata and can be stored differently depending on the DBMS. Here, we will see how the DBMS MySQL stores this metadata, what exactly this data is and how it is possible to extract it.

Let us start by explaining how this data is stored. MySQL uses a very simple system to store information about these schemas. This data is simply stored in a set of tables present in a database named "INFORMATION_SCHEMA" which is present in each instance of MySQL. The following table[1] presents the tables that this database contains as well as a description of its contents.

[1] all information comes from the official website of MySQL[2]

Table Name	Description
CHARACTER_SETS	Provides information about available character sets.
COLLATIONS	Provides information about collations for each character set.
COLLATION_CHARACTER_SET_APPLICABILITY	Indicates what character set is applicable for what collation.
COLUMNS	provides information about columns in tables.
COLUMN_PRIVILEGES	provides information about column privileges.
COLUMN_STATISTICS	provides access to histogram statistics for column values.
ENGINES	provides information about storage engines.
EVENTS	provides information about scheduled events
FILES	provides information about the files in which MySQL tablespace data is stored.
KEYWORDS	lists the words considered keywords by MySQL and, for each one, indicates whether it is reserved.
KEY_COLUMN_USAGE	describes which key columns have constraints.
OPTIMIZER_TRACE	provides information produced by the optimizer tracing capability.
PARAMETERS	provides information about stored procedure and function parameters, and about return values for stored functions.
PARTITIONS	provides information about table partitions.
PLUGINS provides	information about server plugins.
PROCESSLIST	provides information about which threads are running.
PROFILING	provides statement profiling information.
REFERENTIAL_CONSTRAINTS	provides information about foreign keys.
RESOURCE_GROUPS	provides access to information about resource groups.
ROUTINES	provides information about stored routines (both procedures and functions).
SCHEMATA	provides information about databases.
SCHEMA_PRIVILEGES	provides information about schema (database) privileges.
STATISTICS	provides information about table indexes.
ST_GEOMETRY_COLUMNS	provides information about table columns that store spatial data.
ST_SPATIAL_REFERENCE_SYSTEMS	provides information about available spatial reference systems for spatial data.
TABLESPACES	provides information about active tablespaces.
TABLE_CONSTRAINTS	describes which tables have constraints.
TABLE_PRIVILEGES	provides information about table privileges.
TABLES	provides information about tables in databases.
TRIGGERS	provides information about triggers.
USER_PRIVILEGES	provides information about global privileges.
VIEWS	provides information about views in databases.
VIEW_ROUTINE_USAGE	provides access to information about stored functions used in view definitions. (The table does not list information about SQL functions or user-defined functions used in the definitions.)
VIEW_TABLE_USAGE	provides access to information about tables and views used in view definitions.

Table 6.1: MySQL metadata tables

The totality of these tables will not necessarily be useful for us to solve the problem. We will therefore only detail the "COLUMNS" and "KEY_COLUMN_USAGE" tables.

The "COLUMNS" table is composed of 19 columns. With this table it is possible for us to retrieve all the information we need about the column structure of all the tables on which our reasoning engine must work. The following diagram shows the structure of this table in detail. The name of the columns being quite explicit on their content, we will not develop them here.

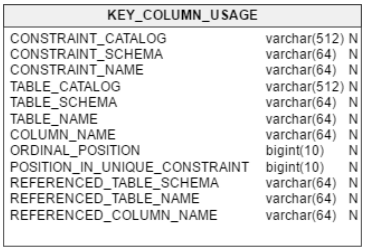
COLUMNS		
TABLE_CATALOG	varchar(512)	N
TABLE_SCHEMA	varchar(64)	N
TABLE_NAME	varchar(64)	N
COLUMN_NAME	varchar(64)	N
ORDINAL_POSITION	bigint(21)	N
COLUMN_DEFAULT	longtext	N
IS_NULLABLE	varchar(3)	N
DATA_TYPE	varchar(64)	N
CHARACTER_MAXIMUM_LENGTH	bigint(21)	N
CHARACTER_OCTET_LENGTH	bigint(21)	N
NUMERIC_PRECISION	bigint(21)	N
NUMERIC_SCALE	bigint(21)	N
CHARACTER_SET_NAME	varchar(32)	N
COLLATION_NAME	varchar(32)	N
COLUMN_TYPE	longtext	N
COLUMN_KEY	varchar(3)	N
EXTRA	varchar(27)	N
PRIVILEGES	varchar(80)	N
COLUMN_COMMENT	varchar(1024)	N

Figure 6.1: INFORMATION_SCHEMA.COLUMNS Table taken from [15]

For our problem, we will use only the following 8 columns:

- COLUMN_NAME
- COLUMN_TYPE
- CHARACTER_SET_NAME
- NUMERIC_PRECISION
- NUMERIC_SCALE
- COLUMN_DEFAULT
- COLUMN_KEY
- TABLE_NAME

The column "KEY_COLUMN_USAGE" is composed of 12 columns allowing to recover all the information concerning the constraints to implement in the DBMS. These constraints can be primary keys, uniqueness constraints and foreign keys. The following diagram shows the structure of this table in detail.



KEY_COLUMN_USAGE		
CONSTRAINT_CATALOG	varchar(512)	N
CONSTRAINT_SCHEMA	varchar(64)	N
CONSTRAINT_NAME	varchar(64)	N
TABLE_CATALOG	varchar(512)	N
TABLE_SCHEMA	varchar(64)	N
TABLE_NAME	varchar(64)	N
COLUMN_NAME	varchar(64)	N
ORDINAL_POSITION	bigint(10)	N
POSITION_IN_UNIQUE_CONSTRAINT	bigint(10)	N
REFERENCED_TABLE_SCHEMA	varchar(64)	N
REFERENCED_TABLE_NAME	varchar(64)	N
REFERENCED_COLUMN_NAME	varchar(64)	N

Figure 6.2: INFORMATION_SCHEMA.KEY_COLUMN_USAGE Table taken from [15]

For our problem, we will use only the following columns:

- TABLE_NAME
- COLUMN_NAME
- COLUMN_NAME
- CONSTRAINT_NAME
- REFERENCED_TABLE_NAME
- REFERENCED_COLUMN_NAME

Finally, let us see how we can extract this data with Java code. The following piece of code is one of the manufacturers of the "Table" Class that you can find in Figure 5.1.

```
/**
*
* @param con Database connection object
* @param name Name of the table to load
* @throws LoadUnexistentTableException If there is no table
*      corresponding to the "name" parameter in the DB to which
*      the "con" parameter is linked
*
**/

public  Table(String name,Connection con) throws
LoadUnexistentTableException {
    try{

        String[] ONE_PARAMETER_TYPE={"YEAR","CHAR","VARCHAR"};
        String[] TWO_PARAMETER_TYPE={"FLOAT","DOUBLE","DECIMAL"};

        foreignKeys = new ArrayList<ForeignKey>();
        Tablecolumn = new ArrayList<Column>();
        this.name = name;

/*************************************************************
*                   Step 1: load Table columns data          *
*************************************************************/

        create Tablecolumn
        SQLSelectQuery select = new SQLSelectQuery(
                        new String[]{"information_schema.columns"},
                con,
                new String[]{"column_name",
                        "column_type",
                        "CHARACTER_SET_NAME",
                        "NUMERIC_PRECISION",
                        "NUMERIC_SCALE",
                        "COLUMN_DEFAULT"},
                "table_name='"+name+"'");

        /*********************************************************
        *Execute the following Query :                          *
        *"Select column_name,column_type,CHARACTER_SET_NAME,     *
        *        NUMERIC_PRECISION, NUMERIC_SCALE,COLUMN_DEFAULT *
        * From information_schema.columns                       *
        * Where table_name=$name;"                              *
        * in this query $name is the parameter "name"           *
        *********************************************************/

        ResultSet rs = select.sqlQueryDo();
        while(rs.next()){
                String colName = rs.getString("column_name");
```

```
            String colType = rs.getString("column_type");
            String charset = rs.getString("CHARACTER_SET_NAME");
            String defaultV = rs.getString("COLUMN_DEFAULT");
            /*
            specific case when for example whe store a FLoat
            without prescice the parameters
            */
            String numPres =
                    rs.getString("NUMERIC_PRECISION")!=null ?
                rs.getString("NUMERIC_PRECISION") :"0";
            String numScal =
                    rs.getString("NUMERIC_SCALE")!=null ?
                rs.getString("NUMERIC_SCALE") :"0";

            if(!colType.contains("(") &&
                    isIn(colType, TWO_PARAMETER_TYPE)){
                colType = String.format("%s(%s,%s)",
                    rs.getString("column_type"),numPres,numScal);
            }else if(!colType.contains("(") &&
                    isIn(colType, ONE_PARAMETER_TYPE) ){
                colType = String.format("%s(%s)",
                    rs.getString("column_type"),numPres);
            }
            this.addColumn(
                    new Column(colName, colType,charset,defaultV)
            );
        }
    rs.close();

    /*************************************************************
    *                      Step 2: load foreign keys             *
    *************************************************************/

    SQLSelectQuery select2 = new SQLSelectQuery(
        new String[]{"INFORMATION_SCHEMA.KEY_COLUMN_USAGE"},
        con,
        new String[]{
            "TABLE_NAME",
            "COLUMN_NAME",
            "CONSTRAINT_NAME",
            "REFERENCED_TABLE_NAME",
            "REFERENCED_COLUMN_NAME"
            },
      "REFERENCED_TABLE_NAME IS NOT NULL AND TABLE_NAME='"+name+"'"
    );

    /*************************************************************
    *Execute the following Query :                               *
    *"Select TABLE_NAME,COLUMN_NAME,CONSTRAINT_NAME*
    *         REFERENCED_TABLE_NAME,REFERENCED_COLUMN_NAME       *
```

```
        * From INFORMATION_SCHEMA.KEY_COLUMN_USAGE                    *
        * Where REFERENCED_TABLE_NAME IS NOT NULL                      *
        *         AND TABLE_NAME=$name;"                               *
        * in this query $name is the parameter "name"                  *
        ***************************************************************/

        ResultSet resultfk = select2.sqlQueryDo();
        while(resultfk.next()){
            foreignKeys.add(
                new ForeignKey(
                        resultfk.getString("REFERENCED_TABLE_NAME"),
                    resultfk.getString("REFERENCED_COLUMN_NAME"),
                    resultfk.getString("COLUMN_NAME"),
                    resultfk.getString("TABLE_NAME"),
                    resultfk.getString("CONSTRAINT_NAME")
                )
            );
        }

        /***************************************************************
        *                    Step 3: load Primary key                  *
        ***************************************************************/

        SQLSelectQuery select3 = new SQLSelectQuery(
                new String[]{"INFORMATION_SCHEMA.COLUMNS"},
                con,
                new String[]{"COLUMN_NAME"},
                "TABLE_NAME = '"+name+"' AND COLUMN_KEY = 'PRI'"
                );
        primaryKey=null;

        /***************************************************************
        *Execute the following Query :                                 *
        *"Select COLUMN_NAME                                          *
        * From information_schema.columns                             *
        * Where TABLE_NAME=$name AND COLUMN_KEY = 'PRI';"        *
        * in this query $name is the parameter "name"                  *
        ***************************************************************/

        ResultSet pri = select3.sqlQueryDo();
        while(pri.next()){
            this.primaryKey = pri.getString("COLUMN_NAME");
        }
    }catch(SQLException e){
    throw new LoadUnexistentTableException(
        "It's impossible to load the table it could doesn't exist"
    );
    }
}
```

This piece of code can be divided into three key steps.

First, it extracts all the information concerning the columns: "their names, types, charsets, numeric prescisions, numeric scales, default values". Then it creates a corresponding java object "column" that will be added to the object "Table" in creation.

The second step is the extraction of the data concerning the potential foreign keys, the referential constraints that this table must satisfy. To do this, the following information is extracted: "the name of the table, the name of the column concerned, the name of the constraint, the table referenced by the constraint and the columns". All this information will be used to create a Java "Foreign Key" which will be added later to the object "Table" in creation.

The last step, is the loading of the primary keys of the target table which is then added to the object under construction. Once these three steps are done, we have a structured Java object in which all the information concerning a chosen table is gathered and easily accessible. This will be very useful for the implementation of our reasoning engine.

Do/undo feature

Finally, let us talk about the latest "EasySQL" feature. As said earlier, this library allows to perform SQL queries in java and obviously to execute them or retrieve the script (as a string).

But that is not all because, as seen above[2], adding foreign keys can have strong impacts on the applications that work with it. This impact can be on several levels. It may have no impact, which is the most desirable case, because it does not require any changes to the application; it may be less when it only prevents the realization of certain functionality of the application; or it may be global when it makes impossible the operation of the entire application.

To respond to these two potentially problematic impacts, we added the "sqlQueryUndo()" method to any query. This method will cancel the previous query (when it is possible). This method makes it possible to make a rollback, restore the state of the Database before the execution of the request.

To do this, it is necessary to proceed in two different ways according to the type of request: whether the query has an impact only on the database schema, or whether it affects the data stored in that database.

The first type is the easiest to manage. If the query has only an impact on the schema, it is sufficient to memorize the old state of the schema and to execute the query restoring this state. Requests of this type are as follows:

- Create a table
- Alter a table

[2]see section 3.5

The second type of request is more delicate, because as it modifies the data present in the database it is also necessary to restore them. To do this, before executing one of these requests, the data concerned by the modification must be loaded and saved for possible restoration. The requests concerned are as follows:

- Delete Data
- Delete tables
- Update data
- Insert data
- Drop table

Finally, it is not possible to implement this feature for all queries. For example, undo a "select" query makes no sense.

6.1.2 Context Analyzer

This software object is simply a Java implementation of the algorithm presented during the subsection 5.1.2. It is a set of "if statements"(as presented on the Figure 5.4), based on EasySQL and its meta-data extraction.

It has the particularity to have its own internal representation of its database table that it modifies depending on the transformations that it chooses - or that are dictated by the user - in order to keep a coherence on the transformation that it operates - or that will be operated with a generated SQL script. Thus, it keeps in memory the old transformations that it operated in order to keep its internal representation of the database coherent.

6.1.3 Prodacon2 GUI

As a direct java code manipulation can be complex enough and could require an important amount of time to understand and master it, a GUI was created to facilitate these manipulations by database managers. This GUI was based on the javaFX 2.0 technology and was partially built with the JavaFX Scene Builder tool from Oracle.

It offered a series of features:

1. Configuring the database settings for the connection. This is an essential part to make the application able to get all schema information and possibly perform some direct transformations with a user's authorization. To retrieve this information it needs the database host name, DB name, port number and also an authorized user login and his password (if the user has only read-only access the *Direct Database Manipulation* feature will not be available)

2. Loading a list of candidate foreign keys from a text file where each line is formatted such as:
   foreingKeyTableName :foreingKeyColumnName: ReferencedTableName: ReferencedColumnName

3. Proceeding with a direct database transformation managed by a database manager.

4. Generating a SQL script that has the same impact as a direct database transformation.

5. Performing a fast analysis.

Direct database transformation

After configuring the database settings [3] and loading a list of candidate foreign keys, a user can start a *Direct Database Manipulation* which means that the application will treat successively all imported potential foreign keys and propose a transformation that the application judges as being the best choice for each one of them.

The user is free to change a set of parameters like the transformation type. Each column will be impacted by this transformation (the referenced or the foreign key column) but we give no guarantee if bad settings are performed.

The application displays some other information to help the user choosing the current type of the columns, the list of potential unmatching values between the two tables and the list of the potential cascade Foreign keys.

With the information made available, s/he has several possibilities :

1. Execute the transformation (the Database will be directly impacted).
   If there are some unmatched values the application proposes several choices:

   (a) Set null all unmatched values.

   (b) Delete unmatched values (the full line) on cascade.

   If there are some cascade foreign keys, the transformation is performed on every linked column involved in the modification.

2. Abort the transformation. No transformation will be performed for the current foreign key.

3. Add an equivalent Trigger. This technique allows to maintain the integrity of the database by adding a trigger that will act as a foreign key but without caring if the previous data is respecting this constraint.

Script generator

This feature is very similar to the previous one. It has exactly the same aspect but the difference is that the database is not directly altered. Instead, the application generates an equivalent SQL script that can be executed by the user (it is recommended to execute this script on a transaction to avoid all potential mistakes or database exceptions).
To perform this generation, the context analyzer creates its own database representation and modify it for every intended modification so that every step of the generated script takes into account every previous database alteration. For this reason, this feature does not support concurrent database modifications during the process.

[3] This feature required a full privileges access

Fast Analysis

This feature performs a fast and simple analysis of the transformations that should be executed for adding the candidate foreign key. The application sends a report of all the transformations that the context analyzer will propose.
The list of transformations is fully automated and does not demand any user interaction for the transformations choices. So, this analysis is not entirely reliable but can still be useful for a first look at the impact significance of the foreign key adding or to check the accuracy of the candidate foreign keys.

6.1.4 Remote diagnostic

To be able to make a remote diagnosis we chose to create a new "Diagnostic" java object which is based on the "ContextAnalyser" object described above. They are implemented with the same reasoning engine and therefore make the same transformations choices on the database to allow the foreign keys to be updated. The only difference is that instead of generating "Transformation" java objects, the "Diagnostics" object will generate "Analysis" objects which role is collecting all information on the proposed transformation and on the database in order to generate a report in the JSON format.

The following frame presents an extract from the report of our "remote diagnosis" on the Oscar database. This extract is placed here to give an idea of the general structure of the report the full report is available at the following address: https://drive.google.com/open?id=1JQP_7pK-JykuHJE6yR28rAmB0nYD2Iq3


```
{"proadcon2Diagnostic": [{
      "advisedNewType": "INT(11)",
      "unmatchingUnsigned": "false",
      "encodageMatching": "true",
      "impossibleAdding": "false",
      "foreignKeyCascade": [],
      "advisedTarget": "ForeignKeyTable",
      "fkAlreadyExist": "false",
      "transformationType": "MBT",
      "unmatchingValuesNumber": "0",
      "message": "",
      "foreignKey": {
        "ReferencedTableName": "Facility",
        "ReferencedColumn": "id",
        "ForeingKeyColumn": "facilityId",
        "ForeingKeyTable": "IntegratorControl",
        "ConstraintName": "fk_Constraint_IntegratorControl_Facility_125"
      },
      "ReferenceCascade": [
        {
          "ReferencedTableName": "Facility",
          "ReferencedColumn": "id",
          "ForeingKeyColumn": "facilityId",
          "ForeingKeyTable": "ClientLink",
          "ConstraintName": "ClientLink_ibfk_1"
```

```
        },
        {
          "ReferencedTableName": "Facility",
          "ReferencedColumn": "id",
          "ForeingKeyColumn": "facilityId",
          "ForeingKeyTable": "DigitalSignature",
          "ConstraintName": "DigitalSignature_ibfk_1"
        },
        {
          "ReferencedTableName": "Facility",
          "ReferencedColumn": "id",
          "ForeingKeyColumn": "facilityId",
          "ForeingKeyTable": "HnrDataValidation",
          "ConstraintName": "HnrDataValidation_ibfk_1"
        },
        {
          "ReferencedTableName": "Facility",
          "ReferencedColumn": "id",
          "ForeingKeyColumn": "facilityId",
          "ForeingKeyTable": "IntegratorConsent",
          "ConstraintName": "IntegratorConsent_ibfk_1"
        },
        {
          "ReferencedTableName": "Facility",
          "ReferencedColumn": "id",
          "ForeingKeyColumn": "facilityId",
          "ForeingKeyTable": "IntegratorConsentComplexExitInterview",
          "ConstraintName": "IntegratorConsentComplexExitInterview_ibfk_1"
        },
        {
          "ReferencedTableName": "Facility",
          "ReferencedColumn": "id",
          "ForeingKeyColumn": "facilityId",
          "ForeingKeyTable": "program",
          "ConstraintName": "program_ibfk_1"
        },
        {
          "ReferencedTableName": "Facility",
          "ReferencedColumn": "id",
          "ForeingKeyColumn": "facility_id",
          "ForeingKeyTable": "room",
          "ConstraintName": "FK_room_facility"
        }
      ],
"dicoTable": [{
      "name": "billingstatus_types",
      "primaryKey": "billingstatus",
      "Tablecolumn": [
        {
          "columnName": "billingstatus",
```

```
      "columnType": "char(1)",
      "defaultValue": "",
      "charset": "latin1",
      "unsigned": "false"
    },
    {
      "columnName": "displayName",
      "columnType": "varchar(20)",
      "defaultValue": "",
      "charset": "latin1",
      "unsigned": "false"
    },
    {
      "columnName": "displayNameExt",
      "columnType": "varchar(50)",
      "defaultValue": "null",
      "charset": "latin1",
      "unsigned": "false"
    },
    {
      "columnName": "sortOrder",
      "columnType": "int(10) unsigned",
      "defaultValue": "0",
      "charset": "null",
      "unsigned": "true"
    },
    {
      "columnName": "billingstatus",
      "columnType": "char(1)",
      "defaultValue": "",
      "charset": "latin1",
      "unsigned": "false"
    },
    {
      "columnName": "displayName",
      "columnType": "varchar(20)",
      "defaultValue": "",
      "charset": "latin1",
      "unsigned": "false"
    },
    {
      "columnName": "displayNameExt",
      "columnType": "varchar(50)",
      "defaultValue": "null",
      "charset": "latin1",
      "unsigned": "false"
    },
    {
      "columnName": "sortOrder",
      "columnType": "int(10) unsigned",
```

```
        "defaultValue": "0",
        "charset": "null",
        "unsigned": "true"
      }
    ],
    "foreignKeys": []}}
```

6.2 Program adaptation

6.2.1 Trigger logger

As said in subsection 5.2.1, the trigger logger mechanism is quite simple. All it requires before adding the different triggers is to create the log-table that records all the simulated foreign key violations.

We developed a solution in JAVA that takes as input a candidate foreign key and returns a Mysql script that has to be executed in order to add the log-table and the triggers in the database.

The SQL code below correspond to the generated script for this implicit foreign key[4] :

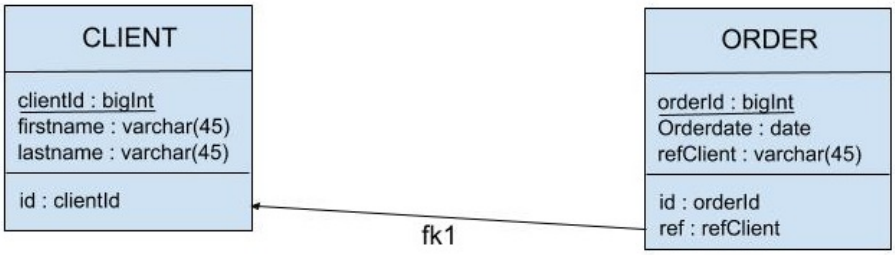


Figure 6.3: Implicit Foreign key FK1

[4]Figure 6.3 represents the foreign key as if it was implemented in the DBMS for demonstrative purpose only.

```
/*****************************************************************
* Creation of the log-table (here called tableTriggerLog)        *
*****************************************************************/

create table IF NOT EXISTS tableTriggerLog (
    foreignKeyName varchar(100) NOT NULL,
    foreignKeyTable varchar(100) NOT NULL,
    foreignKeyColumn varchar(100) NOT NULL,
    referencedTable varchar(100) NOT NULL,
    referencedColumn varchar(100) NOT NULL,
    problemAction varchar(40) NOT NULL,
    dateAndTime timeStamp NOT NULL DEFAULT now()
);

/*****************************************************************
* Creation of the Trigger that will check the inserts in the     *
* Order table                                                    *
*****************************************************************/

CREATE TRIGGER triggerInsertFK1
BEFORE INSERT ON Order
FOR EACH ROW
BEGIN
  IF not (new.refClient IN (SELECT clientId from Client))
  THEN

    INSERT INTO tableTriggerLog (foreignKeyName, foreignKeyTable,
    foreignKeyColumn, referencedTable, referencedColumn, problemAction)
    values("FK1", "Order", "refClient",
    "Client", "clientId", "insert");

  END IF;
END

/*****************************************************************
* Creation of the Trigger that will check the deletes in the     *
* Client table                                                   *
*****************************************************************/

CREATE TRIGGER triggerDeleteFK1
BEFORE delete ON Client
FOR EACH ROW
BEGIN
  IF (old.clientId IN (SELECT refClient from Order))
  THEN

    INSERT INTO tableTriggerLog (foreignKeyName, foreignKeyTable,
    foreignKeyColumn, referencedTable, referencedColumn, problemAction)
```

```
    values("FK1", "Order", "refClient",
    "Client", "clientId", "delete");

  END IF;
END

/*****************************************************************
* Creation of the Trigger that will check the updates in the     *
* Order table                                                    *
*****************************************************************/

CREATE TRIGGER triggerUpdateSourceFk1
  BEFORE UPDATE ON Order
  FOR EACH ROW
BEGIN
  IF((old.refClient <> new.refClient) AND
  not (new.refClient IN (SELECT clientId from Client)))
  THEN

    INSERT INTO tableTriggerLog (foreignKeyName, foreignKeyTable,
    foreignKeyColumn, referencedTable, referencedColumn, problemAction)
    values("Fk1", "Order", "clientRef",
    "Client", "clientId", "updateForeignkeyTable");

  END IF;
END

/*****************************************************************
* Creation of the Trigger that will check the updates in the     *
* Client table                                                   *
*****************************************************************/

CREATE TRIGGER triggerUpdateDestFK1
  BEFORE UPDATE ON Client
  FOR EACH ROW
BEGIN
  IF((old.clientId <> new.clientId) AND
  (old.clientId IN (SELECT clientRef from Order)))
  THEN

    INSERT INTO tableTriggerLog (foreignKeyName, foreignKeyTable,
    foreignKeyColumn, referencedTable, referencedColumn, problemAction)
    values("FK1", "Order", "clientRef",
    "Client", "clientId", "updateReferencedTable");

  END IF;
END
```

Let us not forget that these triggers are only temporary. Therefore, SQL instructions to retrieve them should also be developed (and an instruction to clear the log-table as well). However, we judged that it wasn't interesting to present these instructions to the reader.

Additionally, a mechanism to retrieve the information from the log-table should also be developed but once again, we chose to not dwell on that part.

Finally, the name of a trigger could cause a problem if it already exists, even if it is unlikely - here the name of a trigger describes its function (like triggerInsert) coupled with the name of the foreign key (here : fk1). A workaround mechanism has to be developed but we thought this was not a key point of this thesis.

Chapter 7

Case study : OSCAR

In order to test the different developed tools, a case study in a real environment is needed. Luckily, when we developed our solutions during an internship at the University of Victoria, we had at our disposal a very interesting database to work with : the OSCAR database - OSCAR stands for Open Source Clinical Application Resource.

In this chapter, a description of the OSCAR system is given. Then we will explain what different constraints this system has imposed on our thinking. Finally, the results of some of our developed tools running on it will be presented.

7.1 What is OSCAR?

The OSCAR system is an Electronic Medical Record (EMR) system. *An EMR is a digital medical record that either originates from an electronic format or is converted from paper or hard copy to an online version* according to [6].

OSCAR is a Clinical Management System that is distributed all over Canada and is used by hundreds of clinics and hospitals since 2001.

Its main purpose is to reduce the management cost of the clinics but also to improve health service efficiency by integrating the different clinical health data in its network. This network is also very interesting for answering research questions.

As OSCAR is an open source project, a large community of users, developers and service providers are constantly improving it, adapting it for their own needs and sharing their contributions.

For more information about the OSCAR project, the reader can consult the official website [3].

OSCAR is a legacy system (see section section 1.1 for more details about legacy systems). Indeed, the OSCAR database is very large : it contains more than 480 tables and over 18500 lines of DDL code [11]. The enormous issue with this database is that it contains very few explicitly declared foreign keys (*originally, no foreign key at all had been implemented due to the previous use of the older MyISAM database engine provided by MySQL, which does not support foreign keys*, according to *Gobert and Maes* [25]) which implies that all relationships between tables are implicit. Like many legacy systems, OSCAR has lost

its documentation concerning the initial logical schema so it is very difficult to obtain knowledge of the implicit foreign keys it contains but, as said before in this thesis, the work of *Gobert and Maes*[25] allows us to discharge ourselves from the problem of finding them.

Because it is largely used and because of its active community, OSCAR is in constant evolution and new versions of its system are regularly released. We were, for our part, working with the version 697 of OSCAR on "stable" branch[1] that the reader could find on the Bitbucket website[1].

The complexity of the physical schema of its database implies the requirements defined in section 4.1 where we saw that it would be unlikely for experts to decide to make all implicit foreign keys explicit at once. The reasonable way to perform this database transformation incrementally is: adding the safest foreign keys first, which brought us toward the idea of a decision helper tool that would be able to advise the user for this purpose.

Because OSCAR is a real and very large system with a lot of real world issues that can be often found in other systems that it is a very interesting case study for many research questions. For example, its database has already been the case study of former research that were very useful for ours like [25] or [26].

Some of these issues became constraints for our project and in the next section we will go through them.

7.2 Constraints

Even if the main issue of OSCAR is the loss of the logical schema documentation, it was not a constraint for us strictly speaking (even if it was one for previous research as mentioned in the previous section).

On the other hand, the problem of the data privacy (seen in section 4.4.2) and the data accessibility became profound challenges.

7.2.1 Data protection legislation

The subject of the protection of the data is very actual, particularly this year in the European Union with the new GDPR (the Global Data Protection Regulation) active since 25 May 2018 in all 28 member states.

Of course, data protection laws differ between Canada and Europe but, as the GDPR is stricter than the PIPEDA (Personal Information Protection and Electronic Documents Act[9]) in Canada, it is wise for us to design our tools to be accepted under every legislations.

First of all the data inside the OSCAR database are predominantly personal data because according to the definition (Art. 4, §1 GDPR[5]) a personal data is any information relating to an identified or identifiable natural person. Here, the data are medical data of thousands of patients. It is obvious that the clinics and hospitals have to register their name, address, phone number, ... and link them to all the data concerning them.

So, the data protection applies and without the formal consent of each patient, our tools have not the sufficient authorization to treat these data. This

[1] with commit hash : b685eda393de5b95c6d74966b9436b0e2a0142a3

is why during our internship we had access to a test version of the original database. This mock database contained nearly no data at all. But even if we could only test our tools on an empty database, it was still very interesting to have at our disposal a real-life sized physical schema of a database.

The real challenge that we had not foreseen at the beginning of the internship was that we had to design our tools so that the data manipulation was minimal. For example, at first, our database transformation tool displayed mismatching values to the user. If the data shown are private data, it is likely to be forbidden.

A solution had to be found and it was explained in subsection 5.1.3.

7.2.2 Data accessibility

This problem has already been addressed in subsection 4.4.2 but it is an important milestone in our project as we had to change completely the vision of our system. The issue was that the user of our first database transformation tool imperatively had to be the people with access to the database but also be the people experts that will know the domain sufficiently to make the good decisions. But in reality, in the case of OSCAR, the persons with access to the database are not the experts in the domain.

This is the constraint that brought us to design the Remote Diagnostic tool (see subsection 5.1.3).

Nevertheless we think it would be a waste to consider the previous database transformation tool that needed direct access to the database as obsolete because in most cases, the constraint will not be relevant.

7.3 Results

This section presents the results of the remote diagnostic tool, developed earlier in subsection 6.1.4, as well as their analysis. To perform this diagnostic, we based ourselves on the results of Maxime Gobert and Jérôme Maes' 2013 thesis [25]. The latter had iterated the potential implicit foreign keys present in OSCAR at that time, this list is available in appendixe A.1.

Our tool thus produced a JSON file (available partially in appendix B.1). This document therefore provides all the transformations that our reasoning engine recommends for adding the foreign keys mentioned above as well as a description of the database tables that may be affected by this modification.

Let us begin by analyzing the types of transformations that this document suggests. To do this, we made a pie chart (see Figure 7.2) and a table (see Table 7.3) to see the distribution.

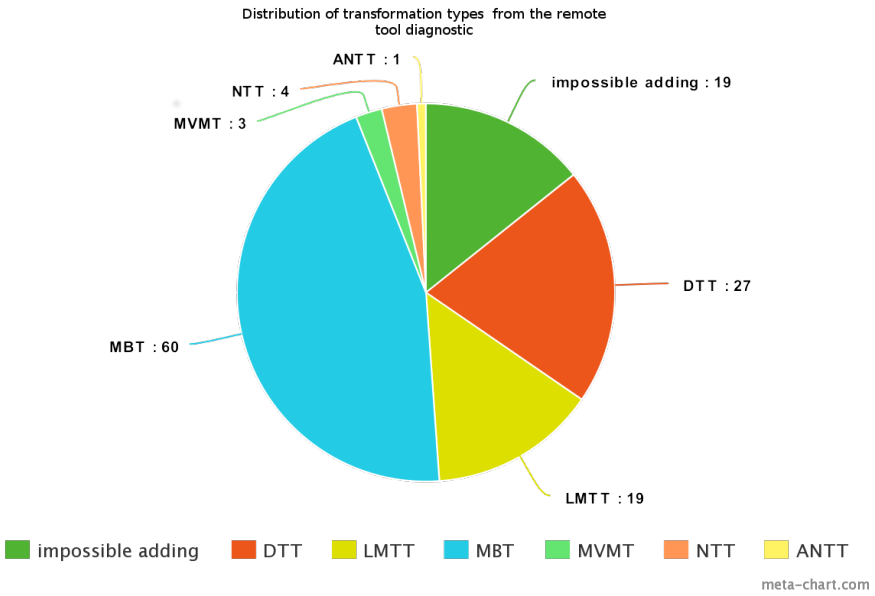


Figure 7.1: Transformation type suggestion

Type of transformation suggested	**Number of cases**	**Unmatching values case** [2]
Time Types Transformation (TTT)	0 (0%)	0
AlphaNumeric Types Transformation (ANTT)	1 (0.8%)	0 (0%)
Matching with Values Mismatching Transformation (MVMT)	3 (2.3%)	3 (100%)
Numeric Types Transformation (NTT)	4 (3.0%)	0 (0%)
Length Mismatch Type Transformation (LMTT)	19 (14.3%)	2 (10.5%)
Impossible adding	19 (14.3%)	/
Different Type Transformation (DTT)	27 (20.3%)	2 (7.4%)
Matching with Basic Transformation (MBT)	60 (45.1%)	0 (0%)

Table 7.1: Type suggested table

These figures are interesting, because they show us that among the 133 foreign keys suggested, our solution is capable of adding 114 (or 85.7%) of them.

[2]Number of times at least one value exists in the database that does not meet the referential constraint.

We will focus on the remaining 14.3% a little later.

This concrete use of our tool also shows us that the typology of transformations developed in the section 4.2.1 must be appropriated, because all cases are found here, with the exception of "Time Types Transformation" (TTT). We can therefore conclude that the typology is well representative of concrete cases in the field.

We can also see that the majority of our cases are "Matching with Basic Transformation" (MBT). This can be reassuring, because, as a reminder, it is the simplest case treated. Here the 2 columns concerned by the foreign key are of the same type because they are both coded on the same number of bits. It is therefore enough to add the constraint in the DBMS to pass this implicit constraint into explicit constraint.

The 3rd column of the table shows us that there are very few cases where there are unmatched values, values that make it impossible to add the foreign key because they are not referenced. This is very good news, however, perhaps because the database on which we tested the tool is a test database and not a real production database and therefore may not be sufficiently populated. To be certain of these results, the test would therefore have to be conducted on an active OSCAR database, which is quite difficult, as these databases are confidential and protected by many private personal information laws.

The other transformations, representing 54 keys or 40.7% are more delicate cases requiring more attention from the user, because there are several alternative ways to add these constraints. Now, let us focus on the transformations that are deemed impossible by our diagnostic tool. These cases are illustrated by Figure 7.2 and Table 7.2.

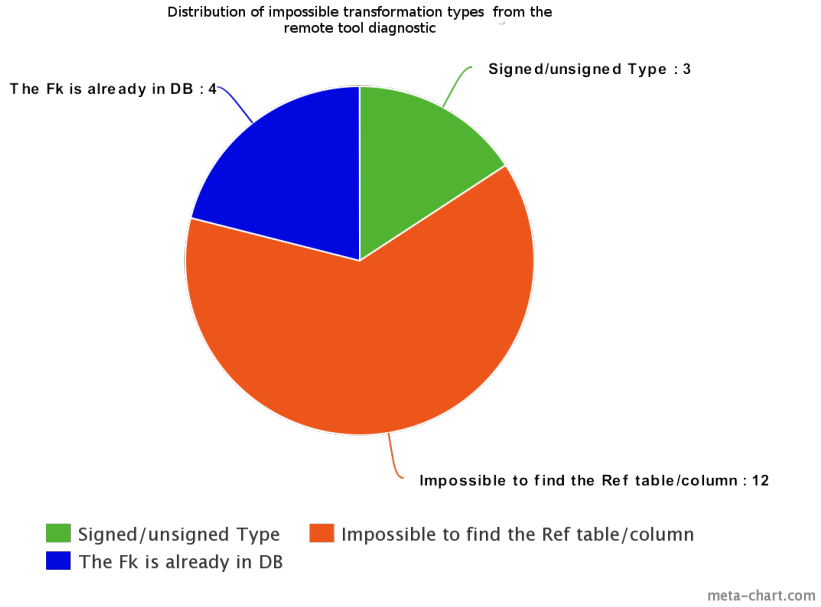


Figure 7.2: Impossible transformation kind

Reason of incapacity	**Percentage**	**Number of cases**
Different signed/unsigned values between the foreign key column and the reference column	15.8 %	3
The Fk is already on the database	21.1 %	4
Impossible to find the Reference table and/or column.	63.2 %	12

Table 7.2: Impossible transformation repartition table

We see here that there are 3 categories of cases that make impossible the addition of a foreign key. First, the case where the type of one of the two columns is a signed type and the type of the other column is an unsigned type. This case could actually be dealt with in our system. This gap is actually a relic of the time when we wanted to achieve a 100% automated system, but we know today that it would be very difficult to achieve. Our solution could thus in its future version take in charge this case.

Then we have the case where we have not been able to find one of the tables or columns concerned by the foreign key. Here, this situation arises because the list of candidate foreign keys that we take as input dates from an older version of OSCAR. This database has in 3 years known a large number of versions and therefore refactors of its schema. After a more detailed analysis, we realized that, most of the time, these tables or columns had been renamed or had been

removed. Finally, the case where the foreign key is already in the database is also due to our out-of-date inputs (the foreign key has been added in the meantime).

This analysis allows us to refine the candidate foreign keys and thus to remove the two cases of impossible addition which are those where the table is not found and where the foreign key has already been added. This gives us the results as illustrated in Figure 7.3 and Table 7.3:

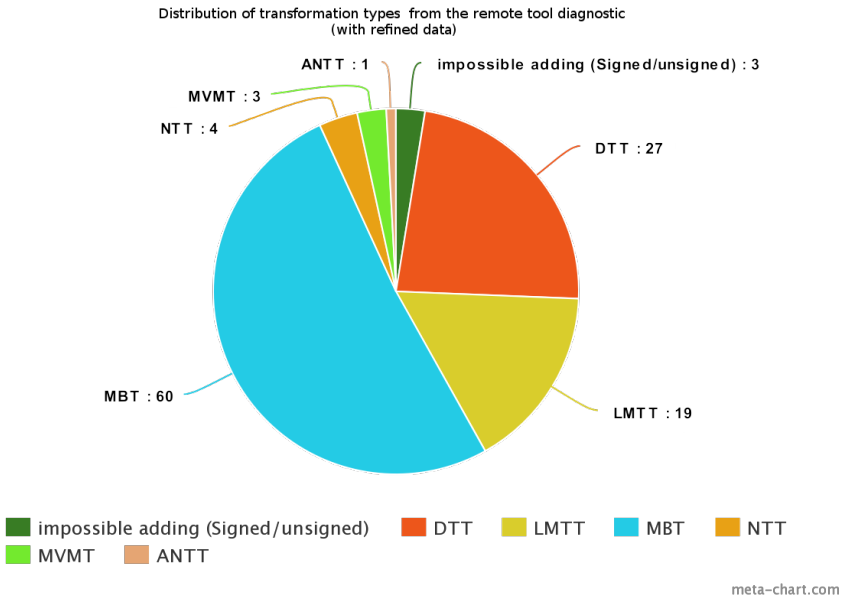


Figure 7.3: Transformation type suggestion (with refined data)

Type of transformation suggested	**Number of case**	**Unmatching values case**
Time Types Transformation (TTT)	0 (0%)	0 (0%)
AlphaNumeric Types Transformation (ANTT)	1 (0.9%)	0 (0%)
Matching with Values Mismatching Transformation (MVMT)	3 (2.5%)	3 (100%)
Impossible adding (signed/unsigned)	3 (2.5%)	/
Numeric Types Transformation (NTT)	4 (3.4%)	0 (0%)
Length Mismatch Type Transformation (LMTT)	19 (16.2%)	2 (10.5%)
Different Type Transformation (DTT)	27 (23%)	2 (7.4%)
Matching with Basic Transformation (MBT)	60 (51.3%)	0 (0%)

Table 7.3: Type suggested table

In conclusion, we see that our system is capable of processing 97.5% of the candidate foreign keys encountered. This result is quite satisfactory while knowing that it would be quite possible to correct the small gap of the remaining 2.5% in a future version. However, it is important to note that this tool only focuses on transformations related to the database schema and the data it contains, but does not take into consideration the resulting application level changes.

Chapter 8

Conclusion

8.1 Summary

In the introduction, the motivation of the thesis has been explained. As new information systems are becoming increasingly rare, a lot of systems are old, very large and have been continuously modified in order to follow a constantly evolving world. These systems, called legacy systems, are characterized by a source code and an architecture that is the product of a long maintaining that can cause some obsolescence, incompleteness and inconsistencies but whose replacement cost would be too high and risky. In these legacy systems, documentation is often missing and at the database level, the conceptual schema can be lost so that unimplemented and undocumented foreign keys - called missing implicit foreign keys - are often lost as well. But evolving a database schema with missing implicit constructs in it, is very perilous for the database integrity so that the recovery of the conceptual schema - and so the reimplementation of these foreign keys - is required. Reverse engineering proposes solutions for retrieving this information but as it is not the subject of our thesis we took the location of the missing foreign keys for granted and we used a tool developed by two students from the University of Namur for that.

So the focus of this thesis was to propose solutions for enforcing foreign keys in legacy systems. It is a difficult problem because implementing a foreign key in a database could require database transformations and also modifications in the application code that is using this database. We then introduced the directions for enforcing foreign keys : a database transformation solution that takes a list of missing foreign keys, treats them one by one and executes all the required transformations, and an application propagation analysis with both a static and a dynamic analysis for determining if the application code still works after adding a foreign key and where changes have to be applied in it.

In chapter 2, we gave a state of the art of different subjects, tools and papers that we used in our thesis or inspired us. We first presented the context in which our thesis was conducted; that is to say inside a collaboration project between the University of Namur and the University of Victoria about the question of managing technical debt. As in this project implicit foreign keys are considered as a form of technical debt, enforcing these foreign keys will greatly help reducing this debt. After that, an introduction to database reverse engineering was given

in order to better understand the description of a tool we used for retrieving the location of implicit foreign keys in a database. We presented also another tool for locating erroneous queries in the application code and a paper about the possible tolerance of inconsistency which helped us design our dynamic analysis.

In chapter 3 we developed more extensively the problem statement of this research. In particular, we defined precisely a foreign key and the scope of an implicit foreign key. The potential issues an implicit foreign key can generate was also explained - loss of these constraints, violation of the logical schema, inconsistency in the database, ... - and our proper definition of "enforcing foreign key" was also detailed. Finally, the impacts the foreign keys enforcement process could have on the application context was explained.

In chapter 4, we established a methodology for elaborating a process answering the problem raised earlier which is a process for enforcing foreign keys in a database. First, we decided which level of automation would be best for this process. We chose a semi-automatic level of automation where the process will act as a decision helper process that, for each implicit foreign key that has to be enforced, will ask the user what to do but, at the same time, will provide him with helpful information that comes from an integrated algorithm that will allow him to make better decisions. After that, in order to design a database transformation algorithm, we constructed a transformation typology to categorized without ambiguity every type of database transformation. Then, a first description of the mechanism of the trigger logger used for the dynamic analysis and motivated by the idea of tolerating inconsistency was given. The principle is to log in a specific table any violation that occurred on a constraint that is not implemented but simulated by triggers. The static analysis using DAHLIA, a tool described in chapter 2, was quickly introduced because, unfortunately, we had not time to develop this part as much as we wanted. Finally, we determined 3 metrics for expressing the dangerousness of implementing a foreign key : the *transformation magnitude* that measures the importance of the database transformation required for implementing the foreign key, the *violation frequency* that measures the probability of errors that would occur after the adding and the *application impact* that measures the importance of the modifications on the application level required after the implementation of the foreign key. We ended this chapter by explaining some problematics we had to face like data privacy issues.

In chapter 5, we designed the different processes and algorithms needed for enforcing foreign keys. For the database transformation part, we first designed an *abstract database manipulation* process that aims to ease readability and manipulability of the software code of our solution by encapsulating database manipulations. Then, the context analyzer, a process that computes the best transformations that should be processed in order to add a given foreign key, is introduced and the algorithm - based on the transformation typology constructed in the previous chapter - that this process uses was also presented. After that, a remote diagnostic process that addresses the problem of the database maintenance and transformation team not necessarily having full access to the database was designed. The idea is to take as input the list of candidate foreign keys and to generate a structured report containing the transformations recommended by the context analyzer process. And in order to answer the data

privacy issue, this report could not contain any sensitive data. For the program adaptation part, the design of the trigger logger was explained, giving the type of information that the log table should retrieve when a violation occurs. Clues for retrieving information from a static analysis was provided afterward. Finally, the last section of this chapter explains how we can use the different processes and tools designed previously in order to give values for each metric defined in chapter 4 that aims to help the user visualize the risk of implementing a certain foreign key.

In chapter 6, we have given the reader the indications about the implementation of every database transformation tool we designed in the previous chapter as well as the implementation of the triggers required for the dynamic analysis. The static analysis and the visualization tool was not implemented.

Finally, in chapter 7, we presented the OSCAR case study (a clinical management system distributed all over Canada), the constraints this case study imposed on our solutions like the data protection legislation or the data accessibility issue, and the results of the remote diagnostic tool on the OSCAR system that allowed us to conclude that our solution is capable of processing 97.5% of the candidate foreign keys encountered which we think is a satisfactory result.

8.2 Evaluation ans future works

Although we have tried to be as comprehensive and as generic as possible, there are a number of elements that should be improved, deepened or developed in the future.

First of all, to better validate our solution, we should test it on a larger database panel, as varied as possible, in order to be sure that we are able to handle as many cases as possible, even some that can be judged to be limited. Indeed, currently we have only tried our tool on a test version of the OSCAR database, which, although relatively large and partially populated, is not a production version that would have been more concrete and representative of the reality.

Another criticism we can make to our solution is that our transformations do not consider some dynamic aspects of DBMS, like triggers for example. It is possible that some foreign keys that we believe implicit are actually implemented by means of these triggers and a future version of our solution could be able to recognize these constructions and could eventually transform them into more conventional foreign keys.

In addition, currently our solution is implemented only for MySQL. It would be interesting to extend this compatibility to a larger number of databases. It should however be noted that the architecture of our solution was thought to allow this with the design pattern "Abstract Factory". But to do this, it would be necessary to study more widely the differences between the main DBMS, mainly on how to extract meta-data and implement a "Factory" for each DBMS.

In addition, in the publication by *Weber, Cleve, Meurice, and Ruiz* subsection 4.4.1 we have developed metrics to calculate the magnitude that the adding of a foreign key could have and the resulting transformation of the schema at

the application level. For that, we based ourselves on 3 metrics which are the number of transformations realized on the database schema and on the values which are stored there, the number of violations of this constraint realized by a dynamic analysis (ex: Logger trigger) and finally the impact that this foreign key could have on the application thanks to a static analysis. Although these 3 criteria are extremely relevant, it would be interesting to discover and study more of them. In [27] the authors mention that having the possibility to determine if the candidate foreign key would be implemented in a critical section of the database or not. Indeed, if we could attribute values of criticality for each part of a database, this could be a very interesting information for the user who will directly know if the candidate foreign key is likely to handle sensible data or not.

Also, the creation of a metric more configurable by a user, which would allow for example to select the foreign keys to add in priority according to the metrics that he considers more important.

Finally, in the implementation of our tools, we mainly focused on the database aspect with schema transformations, content corrections, meta-data analysis... Due to the lack of time, and because it would deserve a thesis dedicated to it, we have developed very few tools to analyze the impact of adding foreign key, and resulting transformations, on applications using this database. This aspect has only been designed at a theoretical level and should be further developed, implemented and tested. It could also be supplemented with an application correction tool that could automatically detect, by static and/or dynamic analysis, which location would pose a problem or represent a risk [1]. For this, it would be possible to rely on reports generated by DAHLIA, the latter is able to analyze the code of an application and notify all the locations of the code communicating with a database and with which tables or columns it does it. It is also able to trace the context of callers which could be useful to determine the sequences of interactions with the database that are no longer possible after adding foreign keys.

[1] see section 3.5 : Foreign keys adding Impact

Bibliography

[1] Bitbucket. See https://bitbucket.org/oscaremr/oscar/branch/stable.

[2] MySQL :: MySQL 8.0 Reference Manual :: 24 INFORMATION_schema Tables | See : https://dev.mysql.com/doc/refman/8.0/en/information-schema.html.

[3] OSCAR EMR | Clinical Management System. See https://oscar-emr.com/.

[4] REVER | see https://www.rever.eu/en.

[5] Règlement (UE) 2016/679 DU PARLEMENT EUROPEEN ET DU CONSEIL du 27 avril 2016.

[6] Techopedia | See https://www.techopedia.com/.

[7] R. Balzer. Tolerating inconsistency [software development]. In *[1991 Proceedings] 13th International Conference on Software Engineering*, pages 158–165, May 1991.

[8] I. D. Baxter and M. Mehlich. Reverse engineering is reverse forward engineering. In *Proceedings of the Fourth Working Conference on Reverse Engineering*, pages 104–113, October 1997.

[9] Office of the Privacy Commissioner of Canada. The Personal Information Protection and Electronic Documents Act (PIPEDA), January 2018.

[10] E. J. Chikofsky and J. H. Cross. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, 7(1):13–17, January 1990.

[11] A. Cleve. Analyzing the Evolution of Data-Intensive Software Systems, 2017.

[12] A. Cleve. Conceptual interpretation of foreign keys, 2017.

[13] Anthony Cleve. Program analysis and transformation for data-intensive system evolution. In *2010 IEEE International Conference on Software Maintenance*, pages 1–6, Timi oara, Romania, September 2010. IEEE.

[14] Anthony Cleve, Maxime Gobert, Loup Meurice, Jerome Maes, and Jens Weber. Understanding database schema evolution: A case study. *Science of Computer Programming*, 97:113–121, January 2015.

[15] Emil Drkušić. vertablo | An Overview of MySQL's Information Schema | See : http://www.vertabelo.com/blog/technical-articles/an-overview-of-mysqls-information-schema, September 2016.

[16] B. Fluri, M. Wursch, and H. C. Gall. Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes. In *14th Working Conference on Reverse Engineering (WCRE 2007)*, pages 70–79, October 2007.

[17] B. Fluri, M. Würsch, E. Giger, and H. C. Gall. Analyzing the co-evolution of comments and source code. Software Quality Journal, 17(4):367–394, 2009.

[18] C. Del Grosso, M. Di Penta, and I. G. de Guzman. An approach for mining services in database oriented applications. In *11th European Conference on Software Maintenance and Reengineering (CSMR'07)*, pages 287–296, March 2007.

[19] Jean-Luc Hainaut. Introduction to Database Reverse Engineering. page 139, 2007.

[20] B. Jose and S. Abraham. Exploring the merits of nosql: A study based on mongodb. In *2017 International Conference on Networks Advances in Computational Technologies (NetACT)*, pages 266–271, July 2017.

[21] Paul Klint. *The software Evolution Volcano, 2011.*

[22] P. Kruchten, R. L. Nord, and I. Ozkaya. Technical Debt: From Metaphor to Theory and Practice. *IEEE Software*, 29(6):18–21, November 2012.

[23] Mario Linares-Vásquez, Boyang Li, Christopher Vendome, and Denys Poshyvanyk. Documenting database usages and schema constraints in database-centric applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis - ISSTA 2016*, pages 270–281, Saarbrücken, Germany, 2016. ACM Press.

[24] Di Lucca, Fasolino, and De Carlini. Recovering class diagrams from data-intensive legacy systems. In *Proceedings 2000 International Conference on Software Maintenance*, pages 52–63, October 2000.

[25] Maxime Gobert and Jérôme Maes. Innovative Techniques and Tools for Database Reverse Engineering in Large Data Intensive Systems, 2013.

[26] C. Nagy, L. Meurice, and A. Cleve. Where was this SQL query executed? a static concept location approach. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 580–584, March 2015.

[27] J. H. Weber, A. Cleve, L. Meurice, and F. J. B. Ruiz. Managing Technical Debt in Database Schemas of Critical Software. In *2014 Sixth International Workshop on Managing Technical Debt*, pages 43–46, September 2014.

Appendix A

External inputs

A.1 Maxime Gobert and Jérôme Maes outputs

The next element represents a series of foreign keys candidate discovered by *Maxime Gobert and Jérôme Maes* in their 2013 thesis [25].

Each line represents a foreign key in the following format:
"TABLE_SOURCE:ATTRIBUT_SOURCE:TABLE_DESTINATION:ATTRIBUT_DESTINATION"

```
billing_history:billingstatus:billingstatus_types:billingstatus
billingmaster:billingstatus:billingstatus_types:billingstatus
billing:billingtype:billingtypes:billingtype
billing_history:billingtype:billingtypes:billingtype
caisi_form_instance_tmpsave:form_id:caisi_form:form_id
cr_policy:role_id:caisi_role:role_id
default_role_access:role_id:caisi_role:role_id
program_access_roles:role_id:caisi_role:role_id
program_provider:role_id:caisi_role:role_id
casemgmt_issue_notes:note_id:casemgmt_note:note_id
casemgmt_tmpsave:note_id:casemgmt_note:note_id
consultationRequestExt:requestId:consultationRequests:requestId
consultdocs:requestId:consultationRequests:requestId
FaxClientLog:requestId:consultationRequests:requestId
consultationRequests:serviceId:consultationServices:serviceId
serviceSpecialists:serviceId:consultationServices:serviceId
caisi_form_data:question_id:cr_securityquestion:question_id
caisi_form_data_tmpsave:question_id:cr_securityquestion:question_id
intake_node:question_id:cr_securityquestion:question_id
intake_node_js:question_id:cr_securityquestion:question_id
survey_test_data:question_id:cr_securityquestion:question_id
caisi_form_instance_tmpsave:user_id:cr_user:user_id
caisi_form_instance:user_id:cr_user:user_id
cr_cert:user_id:cr_user:user_id
cr_policy:user_id:cr_user:user_id
cr_securityquestion:user_id:cr_user:user_id
survey_test_instance:user_id:cr_user:user_id
hl7_msh:message_id:hl7_message:message_id
hl7_pid:message_id:hl7_message:message_id
hl7_obr:pid_id:hl7_pid:pid_id
hl7_orc:pid_id:hl7_pid:pid_id
mdsNTE:segmentID:mdsMSH:segmentID
mdsOBR:segmentID:mdsMSH:segmentID
mdsOBX:segmentID:mdsMSH:segmentID
mdsPID:segmentID:mdsMSH:segmentID
mdsPV1:segmentID:mdsMSH:segmentID
mdsZCL:segmentID:mdsMSH:segmentID
mdsZCT:segmentID:mdsMSH:segmentID
mdsZFR:segmentID:mdsMSH:segmentID
mdsZLB:segmentID:mdsMSH:segmentID
mdsZMC:segmentID:mdsMSH:segmentID
mdsZMN:segmentID:mdsMSH:segmentID
mdsZRG:segmentID:mdsMSH:segmentID
consultationRequests:specId:professionalSpecialists:specId
serviceSpecialists:specId:professionalSpecialists:specId
admission:team_id:program_team:team_id
bed:team_id:program_team:team_id
program_provider:team_id:program_team:team_id
form:provider_no:provider:provider_no
```

billing:provider_no:providerbillcenter:provider_no
billinginr:provider_no:providerbillcenter:provider_no
billingnote:provider_no:providerbillcenter:provider_no
allergies:providerNo:ProviderPreference:providerNo
billing_preferences:providerNo:ProviderPreference:providerNo
CdsClientForm:providerNo:ProviderPreference:providerNo
config_Immunization:providerNo:ProviderPreference:providerNo
consultationRequests:providerNo:ProviderPreference:providerNo
demographicQueryFavourites:providerNo:ProviderPreference:providerNo
DigitalSignature:providerNo:ProviderPreference:providerNo
drugReason:providerNo:ProviderPreference:providerNo
eChart:providerNo:ProviderPreference:providerNo
EyeformConsultationReport:providerNo:ProviderPreference:providerNo
HRMDocumentComment:providerNo:ProviderPreference:providerNo
HRMDocumentToProvider:providerNo:ProviderPreference:providerNo
IntegratorConsent:providerNo:ProviderPreference:providerNo
measurements:providerNo:ProviderPreference:providerNo
measurementsDeleted:providerNo:ProviderPreference:providerNo
MyGroupAccessRestriction:providerNo:ProviderPreference:providerNo
OcanStaffForm:providerNo:ProviderPreference:providerNo
oncall_questionnaire:providerNo:ProviderPreference:providerNo
PageMonitor:providerNo:ProviderPreference:providerNo
PrintResourceLog:providerNo:ProviderPreference:providerNo
ProvPrefApptmentScreenEForm:providerNo:ProviderPreference:providerNo
ProvPrefApptmentScreenForm:providerNo:ProviderPreference:providerNo
ProvPrefApptmentScreenQuickLink:providerNo:ProviderPreference:providerNo
quickListUser:providerNo:ProviderPreference:providerNo
RemoteDataLog:providerNo:ProviderPreference:providerNo
reportByExamples:providerNo:ProviderPreference:providerNo
reportByExamplesFavorite:providerNo:ProviderPreference:providerNo
SecurityToken:providerNo:ProviderPreference:providerNo
form2MinWalk:studyID:rehabStudy2004:studyID
formCaregiver:studyID:rehabStudy2004:studyID
formCESD:studyID:rehabStudy2004:studyID
formCostQuestionnaire:studyID:rehabStudy2004:studyID
formFalls:studyID:rehabStudy2004:studyID
formGripStrength:studyID:rehabStudy2004:studyID
formHomeFalls:studyID:rehabStudy2004:studyID
formInternetAccess:studyID:rehabStudy2004:studyID
formLateLifeFDIFunction:studyID:rehabStudy2004:studyID
formSatisfactionScale:studyID:rehabStudy2004:studyID
formSelfAdministered:studyID:rehabStudy2004:studyID
formSelfEfficacy:studyID:rehabStudy2004:studyID
formSelfManagement:studyID:rehabStudy2004:studyID
formSF36:studyID:rehabStudy2004:studyID
formTreatmentPref:studyID:rehabStudy2004:studyID
IntakeInfo:studyID:rehabStudy2004:studyID
LateLifeFDIDisability:studyID:rehabStudy2004:studyID
SF36Caregiver:studyID:rehabStudy2004:studyID
report_qgviewfield:fieldno:report_filter:fieldno
report_template_criteria:fieldno:report_filter:fieldno
rschedule:sdate:scheduleholiday:sdate
scheduledate:sdate:scheduleholiday:sdate
secObjPrivilege:objectName:secObjectName:objectName
SentToPHRTracking:objectName:secObjectName:objectName
teleplanC12:s21_id:teleplanS21:s21_id
teleplanS00:s21_id:teleplanS21:s21_id
teleplanS22:s21_id:teleplanS21:s21_id
teleplanS23:s21_id:teleplanS21:s21_id
teleplanS25:s21_id:teleplanS21:s21_id
mdsZCL:setID:config_Immunization:setId
mdsZMC:setID:config_Immunization:setId
groupMembers_tbl:groupID:groups_tbl:groupID
msgDemoMap:messageID:messagetbl:messageid
remoteAttachments:messageid:messagetbl:messageid
SecUserRole:role_name:SecRole:role_name
SecObjPrivilege:roleUserGroup:SecRole:role_name
SecObjPrivilege:objectName:SecObjName:objectName
SecObjPrivilege:privilege:SecPrivilege:privilege
HL7HandlerMSHMapping:facility:Facility:id
CdsFormOption:facilityId:Facility:id
ClientLink:facilityId:Facility:id
DemographicContract:facilityId:Facility:id
DigitalSignature:facilityId:Facility:id
HnrDataValidation:facilityId:Facility:id
IntegratorConsent:facilityId:Facility:id
IntegratorControl:facilityId:Facility:id
OcanStaffForm:facilityId:Facility:id
RemoteIntegratedDataCopy:facilityId:Facility:id
caisi_form:facilityId:Facility:id
survey:facilityId:Facility:id

Appendix B

Project output

B.1 remote diagnostic : transformation analysis

The next element is a part of the output generated by the remote diagnostic present in the subsection 6.1.4. This extract contains only the transformations suggested for adding the 133 foreign keys presented in the appendix A.1. So this diagnosis lacks the part including the information relating to the structure of the database schema and tables concerned or impacted by these foreign keys.

The complete diagnosis is available at the following address:
https://drive.google.com/open?id=1JQP_7pK-JykuHJE6yR28rAmB0nYD2Iq3


```
{ "proadcon2Diagnostic" : [ {"advisedNewType" : "CHAR(1)","unmatchingUnsigned" : "false","encodageMatchi
g" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable"
"fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","f
reignKey" : {"ReferencedTableName" : "billingstatus_types", "ReferencedColumn" : "billingstatus", "Foreig
KeyColumn" : "billingstatus", "ForeingKeyTable" : "billing_history", "ConstraintName" : "fk_Constraint_
illing_history_billingstatus_types_0"},"ReferenceCascade" : [ ] },{"advisedNewType" : "CHAR(1)","unmatc
ingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [
], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchi
gValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "billingstatus_types", "Refe
encedColumn" : "billingstatus", "ForeingKeyColumn" : "billingstatus", "ForeingKeyTable" : "billingmaster
, "ConstraintName" : "fk_Constraint_billingmaster_billingstatus_types_1"},"ReferenceCascade" : [ ] },{"
dvisedNewType" : "VARCHAR(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdd
ng" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false"
"transformationType" : "LMTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedT
bleName" : "billingtypes", "ReferencedColumn" : "billingtype", "ForeingKeyColumn" : "billingtype", "Fore
ngKeyTable" : "billing", "ConstraintName" : "fk_Constraint_billing_billingtypes_2"},"ReferenceCascade" :
[ ] },{"advisedNewType" : "VARCHAR(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","imp
ssibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist"
: "false","transformationType" : "LMTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"R
ferencedTableName" : "billingtypes", "ReferencedColumn" : "billingtype", "ForeingKeyColumn" : "billingty
e", "ForeingKeyTable" : "billing_history", "ConstraintName" : "fk_Constraint_billing_history_billingtype
_3"},"ReferenceCascade" : [ ] },{"advisedNewType" : "BIGINT(20)","unmatchingUnsigned" : "false","encoda
eMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignK
yTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message"
: "","foreignKey" : {"ReferencedTableName" : "caisi_form", "ReferencedColumn" : "form_id", "ForeingKeyCo
umn" : "form_id", "ForeingKeyTable" : "caisi_form_instance_tmpsave", "ConstraintName" : "fk_Constraint_c
isi_form_instance_tmpsave_caisi_form_4"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unma
chingUnsigned" : "false","encodageMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" :
[ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "DTT","unmat
hingValuesNumber" : "4","message" : "","foreignKey" : {"ReferencedTableName" : "caisi_role", "Referenced
olumn" : "role_id", "ForeingKeyColumn" : "role_id", "ForeingKeyTable" : "cr_policy", "ConstraintName" :
fk_Constraint_cr_policy_caisi_role_5"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(11)","unmatc
ingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [
], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false","transformationType" : "LMTT","unmatch
ngValuesNumber" : "283","message" : "","foreignKey" : {"ReferencedTableName" : "caisi_role", "Referenced
olumn" : "role_id", "ForeingKeyColumn" : "role_id", "ForeingKeyTable" : "default_role_access", "Constrai
tName" : "fk_Constraint_default_role_access_caisi_role_6"},"ReferenceCascade" : [ ] },{"advisedNewType"
: "BIGINT(20)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","
oreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false","transformation
ype" : "NTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "cais
_role", "ReferencedColumn" : "role_id", "ForeingKeyColumn" : "role_id", "ForeingKeyTable" : "program_acc
ss_roles", "ConstraintName" : "fk_Constraint_program_access_roles_caisi_role_7"},"ReferenceCascade" : [
] },{"advisedNewType" : "BIGINT(20)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossi
```

leAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "
alse","transformationType" : "NTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"Refere
cedTableName" : "caisi_role", "ReferencedColumn" : "role_id", "ForeingKeyColumn" : "role_id", "ForeingKe
Table" : "program_provider", "ConstraintName" : "fk_Constraint_program_provider_caisi_role_8"},"Referenc
Cascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "tru
","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlread
Exist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey"
: {"ReferencedTableName" : "casemgmt_note", "ReferencedColumn" : "note_id", "ForeingKeyColumn" : "note_i
", "ForeingKeyTable" : "casemgmt_issue_notes", "ConstraintName" : "fk_Constraint_casemgmt_issue_notes_ca
emgmt_note_9"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false",
encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "F
reignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","m
ssage" : "","foreignKey" : {"ReferencedTableName" : "casemgmt_note", "ReferencedColumn" : "note_id", "Fo
eingKeyColumn" : "note_id", "ForeingKeyTable" : "casemgmt_tmpsave", "ConstraintName" : "fk_Constraint_ca
emgmt_tmpsave_casemgmt_note_10"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUn
igned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "a
visedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValu
sNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "consultationRequests", "Reference
Column" : "requestId", "ForeingKeyColumn" : "requestId", "ForeingKeyTable" : "consultationRequestExt", "
onstraintName" : "fk_Constraint_consultationRequestExt_consultationRequests_11"},"ReferenceCascade" : [
] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossible
dding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "fal
e","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"Reference
TableName" : "consultationRequests", "ReferencedColumn" : "requestId", "ForeingKeyColumn" : "requestId",
"ForeingKeyTable" : "consultdocs", "ConstraintName" : "fk_Constraint_consultdocs_consultationRequests_12
},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatc
ing" : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTab
e","fkAlreadyExist" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","message" :
"" "foreignKey" : {"ReferencedTableName" : "consultationRequests", "ReferencedColumn" : "requestId", "Forei
gKeyColumn" : "requestId", "ForeingKeyTable" : "FaxClientLog", "ConstraintName" : "fk_Constraint_FaxClie
tLog_consultationRequests_13"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsi
ned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "adv
sedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValues
umber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "consultationServices", "ReferencedC
lumn" : "serviceId", "ForeingKeyColumn" : "serviceId", "ForeingKeyTable" : "consultationRequests", "Cons
raintName" : "fk_Constraint_consultationRequests_consultationServices_14"},"ReferenceCascade" : [ ] },{
advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding
: "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","t
ansformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTable
ame" : "consultationServices", "ReferencedColumn" : "serviceId", "ForeingKeyColumn" : "serviceId", "Fore
ngKeyTable" : "serviceSpecialists", "ConstraintName" : "fk_Constraint_serviceSpecialists_consultationSer
ices_15"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(37)","unmatchingUnsigned" : "false","
ncodageMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "F
reignKeyTable","fkAlreadyExist" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","m
ssage" : "","foreignKey" : {"ReferencedTableName" : "cr_securityquestion", "ReferencedColumn" : "questio
_id", "ForeingKeyColumn" : "question_id", "ForeingKeyTable" : "caisi_form_data", "ConstraintName" : "fk_
onstraint_caisi_form_data_cr_securityquestion_16"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARC
AR(37)","unmatchingUnsigned" : "false","encodageMatching" : "false","impossibleAdding" : "false","foreig
KeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType"
"DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "cr_securit
question", "ReferencedColumn" : "question_id", "ForeingKeyColumn" : "question_id", "ForeingKeyTable" : "
aisi_form_data_tmpsave", "ConstraintName" : "fk_Constraint_caisi_form_data_tmpsave_cr_securityquestion_1
"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(255)","unmatchingUnsigned" : "false","encoda
eMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "Referenc
dTable","fkAlreadyExist" : "false","transformationType" : "LMTT","unmatchingValuesNumber" : "2","message
: "","foreignKey" : {"ReferencedTableName" : "cr_securityquestion", "ReferencedColumn" : "question_id",
"ForeingKeyColumn" : "question_id", "ForeingKeyTable" : "intake_node", "ConstraintName" : "fk_Constraint
intake_node_cr_securityquestion_18"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(255)","unm
tchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" :
[ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false","transformationType" : "LMTT","unma
chingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "cr_securityquestion", "
eferencedColumn" : "question_id", "ForeingKeyColumn" : "question_id", "ForeingKeyTable" : "intake_node_j
", "ConstraintName" : "fk_Constraint_intake_node_js_cr_securityquestion_19"},"ReferenceCascade" : [ ] }
{"advisedNewType" : "VARCHAR(37)","unmatchingUnsigned" : "false","encodageMatching" : "false","impossibl
Adding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "fa
se","transformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"Referenc
dTableName" : "cr_securityquestion", "ReferencedColumn" : "question_id", "ForeingKeyColumn" : "question_
d", "ForeingKeyTable" : "survey_test_data", "ConstraintName" : "fk_Constraint_survey_test_data_cr_securi
yquestion_20"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(64)","unmatchingUnsigned" : "fal
e","encodageMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget"
: "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "
","message" : "","foreignKey" : {"ReferencedTableName" : "cr_user", "ReferencedColumn" : "user_id", "For
ingKeyColumn" : "user_id", "ForeingKeyTable" : "caisi_form_instance_tmpsave", "ConstraintName" : "fk_Con
traint_caisi_form_instance_tmpsave_cr_user_21"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR
64)","unmatchingUnsigned" : "false","encodageMatching" : "false","impossibleAdding" : "false","foreignKe
Cascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "
TT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "cr_user", "Re
erencedColumn" : "user_id", "ForeingKeyColumn" : "user_id", "ForeingKeyTable" : "caisi_form_instance", "
onstraintName" : "fk_Constraint_caisi_form_instance_cr_user_22"},"ReferenceCascade" : [ ] },{"advisedNe
Type" : "VARCHAR(64)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "f
lse","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transfo
mationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName"
"cr_user", "ReferencedColumn" : "user_id", "ForeingKeyColumn" : "user_id", "ForeingKeyTable" : "cr_cert
, "ConstraintName" : "fk_Constraint_cr_cert_cr_user_23"},"ReferenceCascade" : [ ] },{"advisedNewType" :
"VARCHAR(64)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","f
reignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationT
pe" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "cr_us
r", "ReferencedColumn" : "user_id", "ForeingKeyColumn" : "user_id", "ForeingKeyTable" : "cr_policy", "Co
straintName" : "fk_Constraint_cr_policy_cr_user_24"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VA
CHAR(128)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","forg
nKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false","transformationType
: "LMTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "cr_user
, "ReferencedColumn" : "user_id", "ForeingKeyColumn" : "user_id", "ForeingKeyTable" : "cr_securityquesti
n", "ConstraintName" : "fk_Constraint_cr_securityquestion_cr_user_25"},"ReferenceCascade" : [ ] },{"adv
sedNewType" : "VARCHAR(64)","unmatchingUnsigned" : "false","encodageMatching" : "false","impossibleAddin
" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","
ransformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTabl
Name" : "cr_user", "ReferencedColumn" : "user_id", "ForeingKeyColumn" : "user_id", "ForeingKeyTable" : "
urvey_test_instance", "ConstraintName" : "fk_Constraint_survey_test_instance_cr_user_26"},"ReferenceCasc

```
de" : [ ] },{"advisedNewType" : "INT(10) UNSIGNED","unmatchingUnsigned" : "false","encodageMatching" :
true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAl
eadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreign
ey" : {"ReferencedTableName" : "hl7_message", "ReferencedColumn" : "message_id", "ForeingKeyColumn" : "m
ssage_id", "ForeingKeyTable" : "hl7_msh", "ConstraintName" : "fk_Constraint_hl7_msh_hl7_message_27"},"Re
erenceCascade" : [ ] },{"advisedNewType" : "INT(10) UNSIGNED","unmatchingUnsigned" : "false","encodageM
tching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyT
ble","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" :
","foreignKey" : {"ReferencedTableName" : "hl7_message", "ReferencedColumn" : "message_id", "ForeingKeyC
lumn" : "message_id", "ForeingKeyTable" : "hl7_pid", "ConstraintName" : "fk_Constraint_hl7_pid_hl7_messa
e_28"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10) UNSIGNED","unmatchingUnsigned" : "false"
"encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "
oreignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","
essage" : "","foreignKey" : {"ReferencedTableName" : "hl7_pid", "ReferencedColumn" : "pid_id", "ForeingK
yColumn" : "pid_id", "ForeingKeyTable" : "hl7_obr", "ConstraintName" : "fk_Constraint_hl7_obr_hl7_pid_29
},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10) UNSIGNED","unmatchingUnsigned" : "false","enc
dageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "Forei
nKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","messa
e" : "","foreignKey" : {"ReferencedTableName" : "hl7_pid", "ReferencedColumn" : "pid_id", "ForeingKeyCol
mn" : "pid_id", "ForeingKeyTable" : "hl7_orc", "ConstraintName" : "fk_Constraint_hl7_orc_hl7_pid_30"},"R
ferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching"
: "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","f
AlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","fore
gnKey" : {"ReferencedTableName" : "mdsMSH", "ReferencedColumn" : "segmentID", "ForeingKeyColumn" : "segm
ntID", "ForeingKeyTable" : "mdsNTE", "ConstraintName" : "fk_Constraint_mdsNTE_mdsMSH_31"},"ReferenceCasc
de" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","i
possibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExis
" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"
eferencedTableName" : "mdsMSH", "ReferencedColumn" : "segmentID", "ForeingKeyColumn" : "segmentID", "For
ingKeyTable" : "mdsOBR", "ConstraintName" : "fk_Constraint_mdsOBR_mdsMSH_32"},"ReferenceCascade" : [ ]
,{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdd
ng" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false"
"transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTa
leName" : "mdsMSH", "ReferencedColumn" : "segmentID", "ForeingKeyColumn" : "segmentID", "ForeingKeyTable
: "mdsOBX", "ConstraintName" : "fk_Constraint_mdsOBX_mdsMSH_33"},"ReferenceCascade" : [ ] },{"advisedN
wType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "fals
","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transforma
ionType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "
dsMSH", "ReferencedColumn" : "segmentID", "ForeingKeyColumn" : "segmentID", "ForeingKeyTable" : "mdsPID"
"ConstraintName" : "fk_Constraint_mdsPID_mdsMSH_34"},"ReferenceCascade" : [ ] },{"advisedNewType" : "I
T(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignK
yCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" :
MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "mdsMSH", "Re
erencedColumn" : "segmentID", "ForeingKeyColumn" : "segmentID", "ForeingKeyTable" : "mdsPV1", "Constrain
Name" : "fk_Constraint_mdsPV1_mdsMSH_35"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unm
tchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" :
[ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmat
hingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "mdsMSH", "ReferencedColu
n" : "segmentID", "ForeingKeyColumn" : "segmentID", "ForeingKeyTable" : "mdsZCL", "ConstraintName" : "fk
Constraint_mdsZCL_mdsMSH_36"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsig
ed" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advi
edTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesN
mber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "mdsMSH", "ReferencedColumn" : "segme
tID", "ForeingKeyColumn" : "segmentID", "ForeingKeyTable" : "mdsZCT", "ConstraintName" : "fk_Constraint_
dsZCT_mdsMSH_37"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "fals
","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" :
"ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0"
"message" : "","foreignKey" : {"ReferencedTableName" : "mdsMSH", "ReferencedColumn" : "segmentID", "Fore
ngKeyColumn" : "segmentID", "ForeingKeyTable" : "mdsZFR", "ConstraintName" : "fk_Constraint_mdsZFR_mdsMS
_38"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodage
atching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKey
able","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" :
","foreignKey" : {"ReferencedTableName" : "mdsMSH", "ReferencedColumn" : "segmentID", "ForeingKeyColumn
: "segmentID", "ForeingKeyTable" : "mdsZLB", "ConstraintName" : "fk_Constraint_mdsZLB_mdsMSH_39"},"Refe
enceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" :
true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAl
eadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreign
ey" : {"ReferencedTableName" : "mdsMSH", "ReferencedColumn" : "segmentID", "ForeingKeyColumn" : "segment
D", "ForeingKeyTable" : "mdsZMC", "ConstraintName" : "fk_Constraint_mdsZMC_mdsMSH_40"},"ReferenceCascade
: [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impo
sibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist"
"false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"Ref
rencedTableName" : "mdsMSH", "ReferencedColumn" : "segmentID", "ForeingKeyColumn" : "segmentID", "Forein
KeyTable" : "mdsZMN", "ConstraintName" : "fk_Constraint_mdsZMN_mdsMSH_41"},"ReferenceCascade" : [ ] },{
advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding
: "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","t
ansformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTable
ame" : "mdsMSH", "ReferencedColumn" : "segmentID", "ForeingKeyColumn" : "segmentID", "ForeingKeyTable" :
"mdsZRG", "ConstraintName" : "fk_Constraint_mdsZRG_mdsMSH_42"},"ReferenceCascade" : [ ] },{"advisedNewT
pe" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false",
foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformatio
Type" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "pro
essionalSpecialists", "ReferencedColumn" : "specId", "ForeingKeyColumn" : "specId", "ForeingKeyTable" :
consultationRequests", "ConstraintName" : "fk_Constraint_consultationRequests_professionalSpecialists_43
},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatc
ing" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTabl
","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "",
foreignKey" : {"ReferencedTableName" : "professionalSpecialists", "ReferencedColumn" : "specId", "Forein
KeyColumn" : "specId", "ForeingKeyTable" : "serviceSpecialists", "ConstraintName" : "fk_Constraint_servi
eSpecialists_professionalSpecialists_44"},"ReferenceCascade" : [ ] },{"advisedNewType" : "BIGINT(20)","
nmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade
: [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "NTT","un
atchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "program_team", "Refer
ncedColumn" : "team_id", "ForeingKeyColumn" : "team_id", "ForeingKeyTable" : "admission", "ConstraintNam
" : "fk_Constraint_admission_program_team_45"},"ReferenceCascade" : [ ] },{"advisedNewType" : "BIGINT(2
)","unmatchingUnsigned" : "true","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCas
ade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "NTT"
"unmatchingValuesNumber" : "0","message" : "Differents signed/unsigned values between the foreign key co
umn and the reference column","foreignKey" : {"ReferencedTableName" : "program_team", "ReferencedColumn"
```

```
: "team_id", "ForeingKeyColumn" : "team_id", "ForeingKeyTable" : "bed", "ConstraintName" : "fk_Constrain
_bed_program_team_46"},"ReferenceCascade" : [ ] },{"advisedNewType" : "BIGINT(20)","unmatchingUnsigned"
: "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedT
rget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumbe
" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "program_team", "ReferencedColumn" : "tea
_id", "ForeingKeyColumn" : "team_id", "ForeingKeyTable" : "program_provider", "ConstraintName" : "fk_Con
traint_program_provider_program_team_47"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","
nmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade
: [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","un
atchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "provider", "Reference
Column" : "provider_no", "ForeingKeyColumn" : "provider_no", "ForeingKeyTable" : "form", "ConstraintName
: "fk_Constraint_form_provider_48"},"ReferenceCascade" : [ {"ReferencedTableName" : "provider", "Refere
cedColumn" : "provider_no", "ForeingKeyColumn" : "linkProviderNo", "ForeingKeyTable" : "ClientLink", "Co
straintName" : "ClientLink_ibfk_3"}, {"ReferencedTableName" : "provider", "ReferencedColumn" : "provider
no", "ForeingKeyColumn" : "unlinkProviderNo", "ForeingKeyTable" : "ClientLink", "ConstraintName" : "Clie
tLink_ibfk_4"}, {"ReferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColu
n" : "providerNo", "ForeingKeyTable" : "DigitalSignature", "ConstraintName" : "DigitalSignature_ibfk_2"}
{"ReferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "validato
ProviderNo", "ForeingKeyTable" : "HnrDataValidation", "ConstraintName" : "HnrDataValidation_ibfk_3"}, {"
eferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "providerNo",
"ForeingKeyTable" : "IntegratorConsent", "ConstraintName" : "IntegratorConsent_ibfk_3"}, {"ReferencedTab
eName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "provider_no", "ForeingKey
able" : "program_client_restriction", "ConstraintName" : "FK_pcr_provider"} ] },{"advisedNewType" : "VAR
HAR(6)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreign
eyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" :
"MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "providerbil
center", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "provider_no", "ForeingKeyTable" : "bi
ling", "ConstraintName" : "fk_Constraint_billing_providerbillcenter_49"},"ReferenceCascade" : [ ] },{"a
visedNewType" : "VARCHAR(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAddi
g" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false",
transformationType" : "LMTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTa
leName" : "providerbillcenter", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "provider_no",
ForeingKeyTable" : "billinginr", "ConstraintName" : "fk_Constraint_billinginr_providerbillcenter_50"},"R
ferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","encodageMatchi
g" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable"
"fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","f
reignKey" : {"ReferencedTableName" : "providerbillcenter", "ReferencedColumn" : "provider_no", "ForeingK
yColumn" : "provider_no", "ForeingKeyTable" : "billingnote", "ConstraintName" : "fk_Constraint_billingno
e_providerbillcenter_51"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","unmatchingUnsign
d" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advis
dTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNu
ber" : "0","message" : "","foreignKey" : {"ProviderPreference", "ReferencedColum
" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "allergies", "ConstraintName" :
"fk_Constraint_allergies_ProviderPreference_52"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHA
(6)","unmatchingUnsigned" : "true","encodageMatching" : "false","impossibleAdding" : "false","foreignKey
ascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "D
T","unmatchingValuesNumber" : "0","message" : "Differents signed/unsigned values between the foreign key
column and the reference column","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "Referenc
dColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "billing_preferences", "
onstraintName" : "fk_Constraint_billing_preferences_ProviderPreference_53"},"ReferenceCascade" : [ ] },
advisedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAd
ing" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false
,"transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedT
bleName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "
oreingKeyTable" : "CdsClientForm", "ConstraintName" : "fk_Constraint_CdsClientForm_ProviderPreference_54
},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","encodageM
tching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyT
ble","fkAlreadyExist" : "false","transformationType" : "MVMT","unmatchingValuesNumber" : "5","message" :
"","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "For
ingKeyColumn" : "providerNo", "ForeingKeyTable" : "config_Immunization", "ConstraintName" : "fk_Constrai
t_config_Immunization_ProviderPreference_55"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6
","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCas
ade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT"
unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreferenc
", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "consultati
nRequests", "ConstraintName" : "fk_Constraint_consultationRequests_ProviderPreference_56"},"ReferenceCas
ade" : [ ] },{"advisedNewType" : "TEXT","unmatchingUnsigned" : "false","encodageMatching" : "true","imp
ssibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist"
: "false","transformationType" : "ANTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"R
ferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "prov
derNo", "ForeingKeyTable" : "demographicQueryFavourites", "ConstraintName" : "fk_Constraint_demographicQ
eryFavourites_ProviderPreference_57"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","unma
chingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" :
{"ReferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "provider
o", "ForeingKeyTable" : "DigitalSignature", "ConstraintName" : "DigitalSignature_ibfk_2"}, {"ReferencedT
bleName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "linkProviderNo", "Forei
gKeyTable" : "ClientLink", "ConstraintName" : "ClientLink_ibfk_3"}, {"ReferencedTableName" : "provider",
"ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "unlinkProviderNo", "ForeingKeyTable" : "Client
ink", "ConstraintName" : "ClientLink_ibfk_4"}, {"ReferencedTableName" : "provider", "ReferencedColumn" :
"provider_no", "ForeingKeyColumn" : "validatorProviderNo", "ForeingKeyTable" : "HnrDataValidation", "Con
traintName" : "HnrDataValidation_ibfk_3"}, {"ReferencedTableName" : "provider", "ReferencedColumn" : "pr
vider_no", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "IntegratorConsent", "ConstraintName"
"IntegratorConsent_ibfk_3"}, {"ReferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "
oreingKeyColumn" : "provider_no", "ForeingKeyTable" : "program_client_restriction", "ConstraintName" : "
K_pcr_provider"} ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType"
"MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPr
ference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "Dig
talSignature", "ConstraintName" : "fk_Constraint_DigitalSignature_ProviderPreference_58"},"ReferenceCasc
de" : [ ] },{"advisedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","encodageMatching" : "true"
impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyE
ist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" :
{"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "
roviderNo", "ForeingKeyTable" : "drugReason", "ConstraintName" : "fk_Constraint_drugReason_ProviderPrefe
ence_59"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","e
codageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "For
ignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","mes
age" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo
, "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "eChart", "ConstraintName" : "fk_Constraint_eCh
rt_ProviderPreference_60"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(20)","unmatchingUnsi
```

ned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advsedTarget" : "ReferencedTable","fkAlreadyExist" : "false","transformationType" : "LMTT","unmatchingValueNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "EyeformConsultationReport", "ConstraintName" : "fk_Constraint_EyeformConsultationReport_ProviderPreference_61"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(20)","unmatchingUnsigned" : "false","encodageMatching" : "true","impssibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false","transformationType" : "LMTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyTable" : "HRMDocumentComment", "ConstraintName" : "fk_Constraint_HRMDocumentComment_ProviderPreference_62"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(20)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTrget" : "ReferencedTable","fkAlreadyExist" : "false","transformationType" : "LMTT","unmatchingValuesNumbr" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "HRMDocumentToProvider", "ConstrantName" : "fk_Constraint_HRMDocumentToProvider_ProviderPreference_63"},"ReferenceCascade" : [ ] },{"advsedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [{"ReferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "IntegratorConsent", "ConstraintName" : "IntegrtorConsent_ibfk_3"}, {"ReferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "linkProviderNo", "ForeingKeyTable" : "ClientLink", "ConstraintName" : "ClientLink_ibfk_3"}, {"ReferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "unlinkProvierNo", "ForeingKeyTable" : "ClientLink", "ConstraintName" : "ClientLink_ibfk_4"}, {"ReferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "DigitalSignature", "ConstraintName" : "DigitalSignature_ibfk_2"}, {"ReferencedTableName" : "provider", "ReferencedColumn" : "provider_no", "ForeingKeyColumn" : "validatorProviderNo", "ForeingKeyTable" : "HnrDtaValidation", "ConstraintName" : "HnrDataValidation_ibfk_3"}, {"ReferencedTableName" : "provider", "RefrencedColumn" : "provider_no", "ForeingKeyColumn" : "provider_no", "ForeingKeyTable" : "program_client_rstriction", "ConstraintName" : "FK_pcr_provider"}], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "IntegratorConsent", "ConstraintName" : "fk_Constraint_IntegratorConsent_ProiderPreference_64"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "measurements", "ConstraintName" : "fk_Constraint_measurements_ProviderPreference_65"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreferncе", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "measurementsDeleted", "ConstraintName" : "fk_Constraint_measurementsDeleted_ProviderPreference_66"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(20)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreayExist" : "false","transformationType" : "LMTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "MyGroupAccessRestriction", "ConstraintName" : "fk_Constraint_MyGrouAccessRestriction_ProviderPreference_67"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","nmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "OcanStaffForm" "ConstraintName" : "fk_Constraint_OcanStaffForm_ProviderPreference_68"},"ReferenceCascade" : [ ] },{"avisedNewType" : "VARCHAR(40)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAddig" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false", "transformationType" : "LMTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTaleName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "oncall_questionnaire", "ConstraintName" : "fk_Constraint_oncall_questionnaire_Providerreference_69"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false","transformationType" : "LMTT","unmatchingValuesNumber" : "","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "PageMonitor", "ConstraintName" : "fk_Constraint_PageMonitor_ProviderPreference_70"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(10),"unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascde" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false","transformationType" : "LMTT""unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreferenc", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "PrintResouceLog", "ConstraintName" : "fk_Constraint_PrintResourceLog_ProviderPreference_71"},"ReferenceCascade" :] },{"impossibleAdding" : "true","message" : "Impossible to find the Reference table and/or column. It can be not exist","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "ProvPrefApptmentScreenEForm", "ConstrantName" : "fk_Constraint_ProvPrefApptmentScreenEForm_ProviderPreference_72"}},{"impossibleAdding" : "true","message" : "Impossible to find the Reference table and/or column. It can be not exist","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "ProvPrefApptmentScreenQuickLink", "ConstraintName" : "fk_Constraint_ProvPrefApptmentScreenQuickLink_ProvidrPreference_74"}},{"advisedNewType" : "VARCHAR(20)","unmatchingUnsigned" : "false","encodageMatching" : true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAleadyExist" : "false","transformationType" : "LMTT","unmatchingValuesNumber" : "0","message" : "","foreigKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColun" : "providerNo", "ForeingKeyTable" : "quickListUser", "ConstraintName" : "fk_Constraint_quickListUser_roviderPreference_75"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(255)","unmatchingUnsigne" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advise Target" : "ReferencedTable","fkAlreadyExist" : "false","transformationType" : "LMTT","unmatchingValuesNuber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColum" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "RemoteDataLog", "ConstraintNam" : "fk_Constraint_RemoteDataLog_ProviderPreference_76"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foeignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationTye" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProvidrPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : reportByExamples", "ConstraintName" : "fk_Constraint_reportByExamples_ProviderPreference_77"},"Referenceascade" : [ ] },{"advisedNewType" : "VARCHAR(6)","unmatchingUnsigned" : "false","encodageMatching" : "tue","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlre

dyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKe
" : {"ReferencedTableName" : "ProviderPreference", "ReferencedColumn" : "providerNo", "ForeingKeyColumn"
: "providerNo", "ForeingKeyTable" : "reportByExamplesFavorite", "ConstraintName" : "fk_Constraint_report
yExamplesFavorite_ProviderPreference_78"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(10)",
unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascad
" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExist" : "false","transformationType" : "LMTT","
nmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "ProviderPreference
 "ReferencedColumn" : "providerNo", "ForeingKeyColumn" : "providerNo", "ForeingKeyTable" : "SecurityToke
", "ConstraintName" : "fk_Constraint_SecurityToken_ProviderPreference_79"},"ReferenceCascade" : [ ] },{
advisedNewType" : "INT(10)","unmatchingUnsigned" :"false","encodageMatching" : "false","impossibleAddin
" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","
ransformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTabl
Name" : "rehabStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyTab
e" : "form2MinWalk", "ConstraintName" : "fk_Constraint_form2MinWalk_rehabStudy2004_80"},"ReferenceCascad
" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "false","im
ossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist
 : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"R
ferencedTableName" : "rehabStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn" : "studyID", "
oreingKeyTable" : "formCaregiver", "ConstraintName" : "fk_Constraint_formCaregiver_rehabStudy2004_81"},"
eferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching
 : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable",
fkAlreadyExist" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","fo
eignKey" : {"ReferencedTableName" : "rehabStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn"
: "studyID", "ForeingKeyTable" : "formCESD", "ConstraintName" : "fk_Constraint_formCESD_rehabStudy2004_8
"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMat
hing" : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTa
le","fkAlreadyExist" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "
","foreignKey" : {"ReferencedTableName" : "rehabStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyCo
umn" : "studyID", "ForeingKeyTable" : "formCostQuestionnaire", "ConstraintName" : "fk_Constraint_formCos
Questionnaire_rehabStudy2004_83"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingU
signed" : "false","encodageMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ],
advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "DTT","unmatchingVa
uesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "rehabStudy2004", "ReferencedCol
mn" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyTable" : "formFalls", "ConstraintName" : "fk
Constraint_formFalls_rehabStudy2004_84"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unma
chingUnsigned" : "false","encodageMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" :
[ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "DTT","unmat
hingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "rehabStudy2004", "Refere
cedColumn" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyTable" : "formGripStrength", "Constra
ntName" : "fk_Constraint_formGripStrength_rehabStudy2004_85"},"ReferenceCascade" : [ ] },{"advisedNewTy
e" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "false","impossibleAdding" : "false",
foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformatio
Type" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "reh
bStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyTable" : "formHo
eFalls", "ConstraintName" : "fk_Constraint_formHomeFalls_rehabStudy2004_86"},"ReferenceCascade" : [ ] }
{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "false","impossibleAdd
ng" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false"
"transformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTa
leName" : "rehabStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyT
ble" : "formInternetAccess", "ConstraintName" : "fk_Constraint_formInternetAccess_rehabStudy2004_87"},"R
ferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching"
: "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","
kAlreadyExist" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","for
ignKey" : {"ReferencedTableName" : "rehabStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn"
 "studyID", "ForeingKeyTable" : "formLateLifeFDIFunction", "ConstraintName" : "fk_Constraint_formLateLif
FDIFunction_rehabStudy2004_88"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUns
gned" : "false","encodageMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "a
visedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "DTT","unmatchingValu
sNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "rehabStudy2004", "ReferencedColum
" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyTable" : "formSatisfactionScale", "ConstraintN
me" : "fk_Constraint_formSatisfactionScale_rehabStudy2004_89"},"ReferenceCascade" : [ ] },{"advisedNewT
pe" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "false","impossibleAdding" : "false"
foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformati
nType" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "re
abStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyTable" : "formS
lfAdministered", "ConstraintName" : "fk_Constraint_formSelfAdministered_rehabStudy2004_90"},"ReferenceCa
cade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "false"
impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyE
ist" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" :
{"ReferencedTableName" : "rehabStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn" : "studyID
, "ForeingKeyTable" : "formSelfEfficacy", "ConstraintName" : "fk_Constraint_formSelfEfficacy_rehabStudy2
04_91"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encoda
eMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "Foreign
eyTable","fkAlreadyExist" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","message
 : "","foreignKey" : {"ReferencedTableName" : "rehabStudy2004", "ReferencedColumn" : "studyID", "Foreing
eyColumn" : "studyID", "ForeingKeyTable" : "formSelfManagement", "ConstraintName" : "fk_Constraint_formS
lfManagement_rehabStudy2004_92"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUn
igned" : "false","encodageMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "
dvisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "DTT","unmatchingVal
esNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "rehabStudy2004", "ReferencedColu
n" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyTable" : "formSF36", "ConstraintName" : "fk_C
nstraint_formSF36_rehabStudy2004_93"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatch
ngUnsigned" : "false","encodageMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" : [
], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "DTT","unmatchi
gValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "rehabStudy2004", "Reference
Column" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyTable" : "formTreatmentPref", "Constrain
Name" : "fk_Constraint_formTreatmentPref_rehabStudy2004_94"},"ReferenceCascade" : [ ] },{"impossibleAdd
ng" : "true","message" : "Impossible to find the Reference table and/or column. It can be not exist","fo
eignKey" : {"ReferencedTableName" : "rehabStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn"
: "studyID", "ForeingKeyTable" : "IntakeInfo", "ConstraintName" : "fk_Constraint_IntakeInfo_rehabStudy20
4_95"}},{"impossibleAdding" : "true","message" : "Impossible to find the Reference table and/or column.
t can be not exist","foreignKey" : {"ReferencedTableName" : "rehabStudy2004", "ReferencedColumn" : "stud
ID", "ForeingKeyColumn" : "studyID", "ForeingKeyTable" : "LateLifeFDIDisability", "ConstraintName" : "fk
Constraint_LateLifeFDIDisability_rehabStudy2004_96"}},{"impossibleAdding" : "true","message" : "Impossib
e to find the Reference table and/or column. It can be not exist","foreignKey" : {"ReferencedTableName"
 "rehabStudy2004", "ReferencedColumn" : "studyID", "ForeingKeyColumn" : "studyID", "ForeingKeyTable" : "
F36Caregiver", "ConstraintName" : "fk_Constraint_SF36Caregiver_rehabStudy2004_97"}},{"advisedNewType" :
INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreig
KeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType"

"MVMT","unmatchingValuesNumber" : "80","message" : "","foreignKey" : {"ReferencedTableName" : "report_filter", "ReferencedColumn" : "fieldno", "ForeingKeyColumn" : "fieldno", "ForeingKeyTable" : "report_qgviefield", "ConstraintName" : "fk_Constraint_report_qgviewfield_report_filter_98"},"ReferenceCascade" : [},{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "fals","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedableName" : "report_filter", "ReferencedColumn" : "fieldno", "ForeingKeyColumn" : "fieldno", "ForeingKeyable" : "report_template_criteria", "ConstraintName" : "fk_Constraint_report_template_criteria_report_filter_99"},"ReferenceCascade" : [ ] },{"advisedNewType" : "DATE","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "scheduleholiday", "ReferencedColumn" : "sdate", "ForeingKeyColumn" : "sdate", "ForeingKeyTable" : "rschedule", "ConstraintName" : "fk_Constraint_rschedule_scheduleholiday_100"},"ReferenceCascade" : [ ] },{"advisedNewType" : "DATE","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "scheduleholiday", "ReferencedColumn" : "sdate", "ForeingKeColumn" : "sdate", "ForeingKeyTable" : "scheduledate", "ConstraintName" : "fk_Constraint_scheduledate_scedeuleholiday_101"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHAR(100)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTaret" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MVMT","unmatchingValuesNumber" : "171","message" : "","foreignKey" : {"ReferencedTableName" : "secObjectName", "ReferencedColumn" : "objectName", "ForeingKeyColumn" : "objectName", "ForeingKeyTable" : "secObjPrivilege", "ConstraintName" : fk_Constraint_secObjPrivilege_secObjectName_102"},"ReferenceCascade" : [ ] },{"advisedNewType" : "VARCHR(100)","unmatchingUnsigned": "false","encodageMatching" : "true","impossibleAdding" : "false","foreigneyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "LMTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "secObjectNme", "ReferencedColumn" : "objectName", "ForeingKeyColumn" : "objectName", "ForeingKeyTable" : "SentToPHTracking", "ConstraintName" : "fk_Constraint_SentToPHRTracking_secObjectName_103"},"ReferenceCascade" :] },{"advisedNewType" : "INT(10) UNSIGNED","unmatchingUnsigned" : "true","encodageMatching" : "true","mpossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ReferencedTable","fkAlreadyExit" : "false","transformationType" : "LMTT","unmatchingValuesNumber" : "0","message" : "Differents signed unsigned values between the foreign key column and the reference column","foreignKey" : {"ReferencedTablName" : "teleplanS21", "ReferencedColumn" : "s21_id", "ForeingKeyColumn" : "s21_id", "ForeingKeyTable" : "teleplanC12", "ConstraintName" : "fk_Constraint_teleplanC12_teleplanS21_104"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "teleplanS21", "ReferencedColumn" : "s21_id", "ForeingKeyColumn" : "s21_id", "ForeingKeyTable" : "teleplanS00", "ConstraintName" : "fk_Constraint_teleplanS00_teleplanS21_105"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossiblAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencdTableName" : "teleplanS21", "ReferencedColumn" : "s21_id", "ForeingKeyColumn" : "s21_id", "ForeingKeyTale" : "teleplanS22", "ConstraintName" : "fk_Constraint_teleplanS22_teleplanS21_106"},"ReferenceCascade" [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossbleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferncedTableName" : "teleplanS21", "ReferencedColumn" : "s21_id", "ForeingKeyColumn" : "s21_id", "ForeingKeTable" : "teleplanS23", "ConstraintName" : "fk_Constraint_teleplanS23_teleplanS21_107"},"ReferenceCascad" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "true","impssibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReerencedTableName" : "teleplanS21", "ReferencedColumn" : "s21_id", "ForeingKeyColumn" : "s21_id", "ForeinKeyTable" : "teleplanS25", "ConstraintName" : "fk_Constraint_teleplanS25_teleplanS21_108"},"ReferenceCasade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : "false", impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExst" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : "ReferencedTableName" : "config_Immunization", "ReferencedColumn" : "setId", "ForeingKeyColumn" : "setID", "ForeingKeyTable" : "mdsZCL", "ConstraintName" : "fk_Constraint_mdsZCL_config_Immunization_109"},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatching" : false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAreadyExist" : "false","transformationType" : "DTT","unmatchingValuesNumber" : "0","message" : "","foreigKey" : {"ReferencedTableName" : "config_Immunization", "ReferencedColumn" : "setId", "ForeingKeyColumn" "setID", "ForeingKeyTable" : "mdsZMC", "ConstraintName" : "fk_Constraint_mdsZMC_config_Immunization_110},"ReferenceCascade" : [ ] },{"advisedNewType" : "INT(10)","unmatchingUnsigned" : "false","encodageMatcing" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTabl","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "", foreignKey" : {"ReferencedTableName" : "groups_tbl", "ReferencedColumn" : "groupID", "ForeingKeyColumn" "groupID", "ForeingKeyTable" : "groupMembers_tbl", "ConstraintName" : "fk_Constraint_groupMembers_tbl_groups_tbl_111"},"ReferenceCascade" : [ ] },{"advisedNewType" : "MEDIUMINT(9)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "","message" : "","foreignKey" : {"ReferencedTableName" : "messagetbl", "ReferencedColumn" : "messageid", "ForeingKeyColumn" : "messageID", "ForeingKeyTable" : "msgDemoMap", "ConstraintName" : "fk_Constraint_msDemoMap_messagetbl_112"},"ReferenceCascade" : [ ] },{"advisedNewType" : "MEDIUMINT(9)","unmatchingUnsiged" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advidedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNmber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "messagetbl", "ReferencedColumn" : "mssageid", "ForeingKeyColumn" : "messageid", "ForeingKeyTable" : "remoteAttachments", "ConstraintName" : fk_Constraint_remoteAttachments_messagetbl_113"},"ReferenceCascade" : [ ] },{"impossibleAdding" : "true","message" : "Impossible to find the Reference table and/or column. It can be not exist","foreignKey" : "ReferencedTableName" : "SecRole", "ReferencedColumn" : "role_name", "ForeingKeyColumn" : "role_name", "oreingKeyTable" : "SecUserRole", "ConstraintName" : "fk_Constraint_SecUserRole_SecRole_114"}},{"impossibeAdding" : "true","message" : "Impossible to find the Reference table and/or column. It can be not exist","foreignKey" : {"ReferencedTableName" : "SecRole", "ReferencedColumn" : "role_name", "ForeingKeyColumn" : "roleUserGroup", "ForeingKeyTable" : "SecObjPrivilege", "ConstraintName" : "fk_Constraint_SecObjPrivilge_SecRole_115"}},{"impossibleAdding" : "true","message" : "Impossible to find the Reference table and/o column. It can be not exist","foreignKey" : {"ReferencedTableName" : "SecObjName", "ReferencedColumn" : "objectName", "ForeingKeyColumn" : "objectName", "ForeingKeyTable" : "SecObjPrivilege", "ConstraintName" : "fk_Constraint_SecObjPrivilege_SecObjName_116"}},{"impossibleAdding" : "true","message" : "Impossible o find the Reference table and/or column. It can be not exist","foreignKey" : {"ReferencedTableName" : "ecPrivilege", "ReferencedColumn" : "privilege", "ForeingKeyColumn" : "privilege", "ForeingKeyTable" : "ScObjPrivilege", "ConstraintName" : "fk_Constraint_SecObjPrivilege_SecPrivilege_117"}},{"advisedNewType" "INT(11)","unmatchingUnsigned" : "false","encodageMatching" : "false","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationTyp" : "DTT","unmatchingValuesNumber" : "17","message" : "","foreignKey" : {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facility", "ForeingKeyTable" : "HL7HandlerMSHMappin", "ConstraintName" : "fk_Constraint_HL7HandlerMSHMapping_Facility_118"},"ReferenceCascade" : [{"Refere

cedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTab
e" : "ClientLink", "ConstraintName" : "ClientLink_ibfk_1"}, {"ReferencedTableName" : "Facility", "Refere
cedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "DigitalSignature", "Constrain
Name" : "DigitalSignature_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "For
ingKeyColumn" : "facilityId", "ForeingKeyTable" : "HnrDataValidation", "ConstraintName" : "HnrDataValida
ion_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "faci
ityId", "ForeingKeyTable" : "IntegratorConsent", "ConstraintName" : "IntegratorConsent_ibfk_1"}, {"Refer
ncedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTa
le" : "IntegratorConsentComplexExitInterview", "ConstraintName" : "IntegratorConsentComplexExitInterview
ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facility
d", "ForeingKeyTable" : "program", "ConstraintName" : "program_ibfk_1"}, {"ReferencedTableName" : "Facil
ty", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facility_id", "ForeingKeyTable" : "room", "Constra
ntName" : "FK_room_facility"}] },{"impossibleAdding" : "true","message" : "Impossible to find the Refern
nce table and/or column. It can be not exist","foreignKey" : {"ReferencedTableName" : "Facility", "Refer
ncedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "CdsFormOption", "ConstraintN
me" : "fk_Constraint_CdsFormOption_Facility_119"}},{"impossibleAdding" : "true","message" : "The Fk is a
ready on the database","foreignKey" : {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "F
reingKeyColumn" : "facilityId", "ForeingKeyTable" : "ClientLink", "ConstraintName" : "fk_Constraint_Clie
tLink_Facility_120"}},{"impossibleAdding" : "true","message" : "Impossible to find the Reference table a
d/or column. It can be not exist","foreignKey" : {"ReferencedTableName" : "Facility", "ReferencedColumn"
: "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "DemographicContract", "ConstraintName" :
"fk_Constraint_DemographicContract_Facility_121"}},{"impossibleAdding" : "true","message" : "The Fk is a
ready on the database","foreignKey" : {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "F
reingKeyColumn" : "facilityId", "ForeingKeyTable" : "DigitalSignature", "ConstraintName" : "fk_Constrain
_DigitalSignature_Facility_122"}},{"impossibleAdding" : "true","message" : "The Fk is already on the dat
base","foreignKey" : {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn"
"facilityId", "ForeingKeyTable" : "HnrDataValidation", "ConstraintName" : "fk_Constraint_HnrDataValidat
on_Facility_123"}},{"impossibleAdding" : "true","message" : "The Fk is already on the database","foreign
ey" : {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId",
"ForeingKeyTable" : "IntegratorConsent", "ConstraintName" : "fk_Constraint_IntegratorConsent_Facility_12
"}},{"advisedNewType" : "INT(11)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossible
dding" : "false","foreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "fal
e","transformationType" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"Reference
TableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable"
: "IntegratorControl", "ConstraintName" : "fk_Constraint_IntegratorControl_Facility_125"},"ReferenceCasc
de" : [{"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId
, "ForeingKeyTable" : "ClientLink", "ConstraintName" : "ClientLink_ibfk_1"}, {"ReferencedTableName" : "F
cility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "DigitalSigna
ure", "ConstraintName" : "DigitalSignature_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedCol
mn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "HnrDataValidation", "ConstraintName"
: "HnrDataValidation_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingK
yColumn" : "facilityId", "ForeingKeyTable" : "IntegratorConsent", "ConstraintName" : "IntegratorConsent_
bfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityI
", "ForeingKeyTable" : "IntegratorConsentComplexExitInterview", "ConstraintName" : "IntegratorConsentCom
lexExitInterview_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyCo
umn" : "facilityId", "ForeingKeyTable" : "program", "ConstraintName" : "program_ibfk_1"}, {"ReferencedTa
leName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facility_id","ForeingKeyTable" :
"room", "ConstraintName" : "FK_room_facility"}] },{"advisedNewType" : "INT(11)","unmatchingUnsigned" :
false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarg
t" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber"
"0","message" : "","foreignKey" : {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "Fore
ngKeyColumn" : "facilityId", "ForeingKeyTable" : "OcanStaffForm", "ConstraintName" : "fk_Constraint_Ocan
taffForm_Facility_126"},"ReferenceCascade" : [{"ReferencedTableName" : "Facility", "ReferencedColumn" :
"id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "ClientLink", "ConstraintName" : "ClientLin
_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilit
Id", "ForeingKeyTable" : "DigitalSignature", "ConstraintName" : "DigitalSignature_ibfk_1"}, {"Referenced
ableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable"
"HnrDataValidation", "ConstraintName" : "HnrDataValidation_ibfk_1"}, {"ReferencedTableName" : "Facility
, "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "IntegratorConsent",
"ConstraintName" : "IntegratorConsent_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn"
"id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "IntegratorConsentComplexExitInterview", "
onstraintName" : "IntegratorConsentComplexExitInterview_ibfk_1"}, {"ReferencedTableName" : "Facility", "
eferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "program", "ConstraintNa
e" : "program_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColum
" : "facility_id", "ForeingKeyTable" : "room", "ConstraintName" : "FK_room_facility"}] },{"advisedNewTy
e" : "INT(11)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","
oreignKeyCascade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformation
ype" : "MBT","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "Faci
ity", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "RemoteIntegrate
DataCopy", "ConstraintName" : "fk_Constraint_RemoteIntegratedDataCopy_Facility_127"},"ReferenceCascade"
[{"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "F
reingKeyTable" : "ClientLink", "ConstraintName" : "ClientLink_ibfk_1"}, {"ReferencedTableName" : "Facili
y", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "DigitalSignature"
"ConstraintName" : "DigitalSignature_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn"
"id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "HnrDataValidation", "ConstraintName" : "H
rDataValidation_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyCol
mn" : "facilityId", "ForeingKeyTable" : "IntegratorConsent", "ConstraintName" : "IntegratorConsent_ibfk_
"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "
oreingKeyTable" : "IntegratorConsentComplexExitInterview", "ConstraintName" : "IntegratorConsentComplexE
itInterview_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn"
: "facilityId", "ForeingKeyTable" : "program", "ConstraintName" : "program_ibfk_1"}, {"ReferencedTableNa
e" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facility_id","ForeingKeyTable" : "roo
", "ConstraintName" : "FK_room_facility"}] },{"advisedNewType" : "INT(11)","unmatchingUnsigned" : "fals
","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyCascade" : [ ], "advisedTarget" :
"ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MBT","unmatchingValuesNumber" : "0"
"message" : "","foreignKey" : {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKe
Column" : "facilityId", "ForeingKeyTable" : "caisi_form", "ConstraintName" : "fk_Constraint_caisi_form_F
cility_128"},"ReferenceCascade" : [{"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "For
ingKeyColumn" : "facilityId", "ForeingKeyTable" : "ClientLink", "ConstraintName" : "ClientLink_ibfk_1"},
{"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "Fore
ngKeyTable" : "DigitalSignature", "ConstraintName" : "DigitalSignature_ibfk_1"}, {"ReferencedTableName"
"Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "HnrDataV
lidation", "ConstraintName" : "HnrDataValidation_ibfk_1"}, {"ReferencedTableName" : "Facility", "Referen
edColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "IntegratorConsent", "Constrain
Name" : "IntegratorConsent_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "Fo
eingKeyColumn" : "facilityId", "ForeingKeyTable" : "IntegratorConsentComplexExitInterview", "ConstraintN
me" : "IntegratorConsentComplexExitInterview_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedC
lumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "program", "ConstraintName" : "prog
am_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facil

```
ty_id", "ForeingKeyTable" : "room", "ConstraintName" : "FK_room_facility"} ] },{"advisedNewType" : "INT(
1)","unmatchingUnsigned" : "false","encodageMatching" : "true","impossibleAdding" : "false","foreignKeyC
scade" : [ ], "advisedTarget" : "ForeignKeyTable","fkAlreadyExist" : "false","transformationType" : "MB
","unmatchingValuesNumber" : "0","message" : "","foreignKey" : {"ReferencedTableName" : "Facility", "Ref
rencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "survey", "ConstraintName"
 "fk_Constraint_survey_Facility_129"},"ReferenceCascade" : [ {"ReferencedTableName" : "Facility", "Refern
cedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "ClientLink", "ConstraintName
 : "ClientLink_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColu
n" : "facilityId", "ForeingKeyTable" : "DigitalSignature", "ConstraintName" : "DigitalSignature_ibfk_1"}
 {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "For
ingKeyTable" : "HnrDataValidation", "ConstraintName" : "HnrDataValidation_ibfk_1"}, {"ReferencedTableNam
" : "Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "Integ
atorConsent", "ConstraintName" : "IntegratorConsent_ibfk_1"}, {"ReferencedTableName" : "Facility", "Refe
encedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "IntegratorConsentComplexExi
Interview", "ConstraintName" : "IntegratorConsentComplexExitInterview_ibfk_1"}, {"ReferencedTableName" :
"Facility", "ReferencedColumn" : "id", "ForeingKeyColumn" : "facilityId", "ForeingKeyTable" : "program",
"ConstraintName" : "program_ibfk_1"}, {"ReferencedTableName" : "Facility", "ReferencedColumn" : "id", "F
reingKeyColumn" : "facility_id","ForeingKeyTable" : "room", "ConstraintName" : "FK_room_facility"} ] }]
}
```